

IBM

\$9.95
CANADA
\$10.95

os/2 Developer

THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT

Vol. 4 No. 3



**Spotlight on
the OS/2 2.0
Team**



**Printing
with
OS/2 2.0**



**Workplace
Shell
Programming**



Delivering the Power: WATCOM C9.0/386

▶ The Widest Range of 32-bit Intel x86 Platforms

32-bit DOS, 32-bit Windows, OS/2 2.0, AutoCAD ADS

▶ The Industry's Leading Code Optimizer

Advanced global optimizer with new 486 optimizations

▶ The Most Comprehensive Toolset

Debugger, profiler, protected-mode compiler and linker, 32-bit DOS extender with royalty-free run-time, licensed components from Microsoft SDK, and more

▶ The Best Value in 32-Bit Tools: \$895*

Unleash 32-bit Power!

WATCOM C9.0/386 lets you exploit the two key 32-bit performance benefits. The 32-bit flat memory model simplifies memory management and lets applications address beyond the 640K limit. Powerful 32-bit instruction processing delivers a significant speed advantage: typically at least a 2x speedup.

You Get:

- ▶ 100% ANSI and SAA compatible: C9.0/386 passes all Plum Hall Validation Suite tests
- ▶ Extensive Microsoft compatibility simplifies porting of 16-bit code
- ▶ Royalty-free run-time for 32-bit DOS, Windows and OS/2 apps
- ▶ Comprehensive toolset includes debugger, linker, profiler and more
- ▶ DOS extender support for Rational, Phar Lap and Ergo
- ▶ Run-time compatible with WATCOM FORTRAN 77/386

32-bit DOS support includes the DOS/4GW 32-bit DOS extender by Rational Systems with royalty-free runtime license

- ▶ Virtual Memory support up to 32Mb

32-bit Windows support enables development and debugging of true 32-bit GUI applications and DLLs.

- ▶ Includes licensed Microsoft SDK components

32-bit OS/2 2.0 support includes development for multiple target environments including OS/2 2.0, 32-bit DOS and 32-bit Windows

- ▶ Access to full OS/2 2.0 API including Presentation Manager
- ▶ Integrated with IBM Workframe/2 Environment

AutoCAD ADS and **ADI** Development: Everything you need to develop and debug ADS and ADI applications for AutoCAD Release 11

Novell's *Network C* for NLM's SDK includes C/386

The Industry's Choice.

Autodesk, Robert Wenig, Manager, AutoCAD for Windows:

"At Autodesk, we're using WATCOM C/386 in the development of strategic new products since it gives us a competitive edge through early access to new technologies. We also highly recommend WATCOM C/386 to third party AutoCAD add-on (ADS and ADI) developers."

Fox Software, David Fulton, President: "FoxPro 2.0 itself is written in WATCOM C, and takes advantage of its many superior features. Optimizing for either speed or compactness is not uncommon, but to accomplish both was quite remarkable."

GO, Robert Carr, Vice President of Software: "After looking at the 32-bit Intel 80x86 tools available in the industry, WATCOM C was the best choice. Key factors in our decision were performance, functionality, reliability and technical support."

IBM, John Soyring, Director of OS/2 Software Developer Programs: "IBM and WATCOM are working together closely to integrate these compilers with the OS/2 2.0 Programmer's Workbench."

Lotus, David Reed, Chief Scientist and Vice President, Pen-Based Applications: "In new product development we're working with WATCOM C because of superior code optimization, responsive support, and timely delivery of technologies important to us like p-code and support for GO Corp's. PenPoint."

Novell, Nancy Woodward, V.P. and G.M., Development Products: "We searched the industry for the best 386 C compiler technology to incorporate with our developer toolkits. Our choice was WATCOM."



WATCOM C9.0/386

- ▶ 100% ANSI C optimizing compiler
- ▶ Protected-mode compiler ▶ OS/2 hosted-compiler ▶ Royalty-free DOS extender with VMM support ▶ Licensed components of the Microsoft Windows SDK
- ▶ Interactive source-level debugger ▶ Linker
- ▶ Protected-mode linker ▶ OS/2-hosted linker ▶ Profiler
- ▶ Object code librarian ▶ Object code disassembler ▶ MAKE facility ▶ Patch facility ▶ Object module convert utility
- ▶ Windows supervisor ▶ Bind facility for Windows applications
- ▶ 32-bit run-time library object code ▶ Special 32-bit libraries for Windows API ▶ Graphics library for Extended DOS applications ▶ 32-bit Run-time libraries for Windows ▶ 32-bit Run-time libraries for OS/2

Special Offer

Buy **WATCOM C9.0/386** and you'll be eligible to obtain:

WATCOM C9.0 Delta Pack provides you with the tools necessary to develop and debug 16-bit applications for DOS, Windows, and OS/2. **Only \$99.** (\$495 comparative separate cost)

WATCOM FORTRAN 77/386 Delta Pack provides you with everything necessary to develop and debug 32-bit FORTRAN applications for extended DOS, 32-bit Windows and OS/2 2.0. **Only \$399.** (\$895 comparative separate cost)



WATCOM

1-800-265-4555

The Leader in 32-bit Development Tools

415 Phillip Street, Waterloo, Ontario, Canada
Telephone: (519) 886-3700, Fax: (519) 747-4971

*Price does not include freight and taxes where applicable. Authorized dealers may sell for less.
WATCOM C and Lightning Device are trademarks of WATCOM Systems Inc.
DOS/4G and DOS/16M are trademarks of Rational Systems Inc.
Other trademarks are the properties of their respective owners.
Copyright 1992 WATCOM Products Inc.

Spend three days at the '92 Embedded Systems Conference.

Spend the year using what you learned.

Helpful. Inspiring. Top Notch. Excellent. These are just some of the words our past attendees use to describe the Embedded Systems Conference faculty. In just three days you'll find answers to the toughest questions you face presented by leaders in the embedded development community. Faculty who know your job is becoming more complex; who work everyday on real workable solutions to the toughest problems you face. People like Larry Constantine, Steve Mellor, Paul Ward, Derek Hatley, Imtiaz Pirbhai, P.J. Plauger, and Michael Slater.

Build Your Own Program.

Customize your own program by choosing from over 75 lectures and workshops that cover a wide range of issues in embedded development. You'll get top notch advice on real-time programming, CASE and object-oriented methodologies, embedded project management, programming languages, debugging and algorithms and much more.

THE FOURTH ANNUAL EMBEDDED SYSTEMS CONFERENCE

SEPTEMBER 21-24, 1992

SANTA CLARA CONVENTION CENTER
SANTA CLARA, CALIFORNIA



Attendees can choose from over 75 technical sessions led by the leaders in embedded development.

Try Out The Latest Technology At The Embedded Systems Development Products Exhibition.

The Embedded Systems Conference will also be the site of the largest most highly targeted exhibition of embedded development products and services. Nowhere else will you find all the latest tools and utilities on display at one time.



The Santa Clara Convention Center is conveniently located in the heart of Silicon Valley.

You'll Come Away With New Ideas That Translate Directly Into Increased Productivity.

Most importantly, the Embedded Systems Conference is a peer environment that fosters dynamic exchange of vital information, ideas, and insights. You'll find professional excellence throughout the conference; in the lectures and workshops, on the floor of the



Over 120 exhibitors will be showcasing cutting edge tools and utilities.

exposition, in the tutorials, and in informal receptions and hospitality events. And you'll get it all in three days.

Phone us, fax us or mail us today for more information.

TEL: (415) 905-2354
FAX: (415) 905-2220

☐ YES! Please send me more information about the Embedded Systems Conference.

IOS2

Name _____ Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone (____) _____ FAX (____) _____

Send info on: ☐ attending ☐ exhibiting Please also send info to: _____



EMBEDDED
SYSTEMS
CONFERENCE

Mail or FAX to Embedded Systems Conference Miller Freeman Inc.
P.O. Box 7843, San Francisco, CA 94120-7843 — TEL (415) 905-2354; FAX (415) 905-2220

IBM OS/2 Developer

THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT

SUMMER 1992



EDITOR'S COMMENTS

A new OS/2; a new magazine

5



SPOTLIGHT

Spotlight on the OS/2 2.0 Development Team

6



SOFTWARE TOOLS

Building Production-Quality Client/Server Applications

Application Development Aids for OS/2 2.0

The Enhanced Editor

IBM Snapdump/2: Capture File for Problem Determination

16

26

32

34



32-BIT

Application Printing Using OS/2 2.0

Where, Oh Where, Did the Print Manager Go?

DOS Application Printing: Understanding the Differences Under OS/2

Class Objects in SOM

The OS/2 Workplace Programming Interface

42

52

58

67

78



MULTIMEDIA

DVI Support by AudioVisual Connection

Listen to the Sounds: The MMPM/2 Audio Subsystem

90

98



DATABASE

Advanced Programming Considerations for Database Manager

User Exits: Using Nonstandard Devices with DBM

103

110



LAN

The LAN Transport Layer of Extended Services and LAN Services

Creating 32 Bit NetBIOS C Programs

121

124



COMMUNICATIONS

Hyperaccess/5 and OS/2 Multitasking Communications

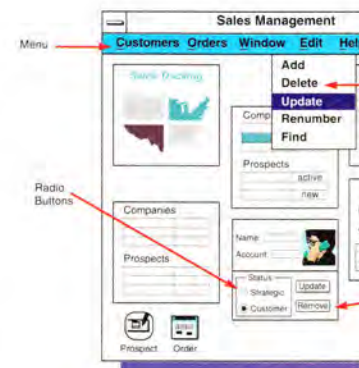
127



SYSTEMS APPLICATION ARCHITECTURE

CCL/2: A User Interface Migration Tool for OS/2 2.0

132



The IBM OS/2 Developer is published quarterly by IBM Software Support, Internal Zip 2230, 1000 N.W. 51st St., Boca Raton, Fla. 33429, (407) 982-6408, fax (407) 443-4233. IBM employees and branch office customers can subscribe to the IBM OS/2 Developer through IBM Mechanicsburg's Systems Library Services (SLSS) using the IBM OS/2 Developer's order number: G362-0001. Others can subscribe by calling (800) WANT-OS2. Subscriptions (U.S. only) are \$39.95 yearly. International subscriptions are also available by phone (708) 647-5960, fax (708) 647-0537.

© Copyright 1992 by International Business Machines Corp. Printed in U.S.A.

EDITOR

Dick Conklin

EXECUTIVE EDITOR

Nicole Freeman

PRODUCTION EDITOR

Peter Altenberg

ASSISTANT COPY/PRODUCTION EDITOR

Lisa Gluskin

PUBLISHER

Regina Starr Ridley

GROUP DIRECTOR

Don Pazour

**NATIONAL SALES MANAGER/
ADVERTISING SALES STAFF**

Cathy Passage

Pam D. Grossman

Michele Anet

(West: 415-905-2392)

**REGIONAL SALES MANAGER/
ADVERTISING SALES STAFF**

Veronica Smith

Jo Ben-Atar

Michele Blake

(East: 212-683-9294)

MARKETING MANAGER

Susan McDonald

MARKETING ASSISTANT

Christopher H. Clarke

**ADVERTISING PRODUCTION
COORDINATOR**

Dede De Leon

DIRECTOR OF PRODUCTION

Andy Mickus

DIRECTOR OF CIRCULATION

Jerry Okabe

CIRCULATION MANAGER

Kathy Henry

SUBSCRIPTION INQUIRIES

U.S.: (800) WANT-OS2

Foreign: (708) 647-5960

IBM OS/2 Developer is published by the Personal Systems Line of Business of International Business Machines Corp., Boca Raton, Fla., U.S.A., Dick Conklin, Editor. Titles and abstracts, but no other portions, of information in this publication may be copied and distributed by computer-based and other information service systems. Permission to republish information from this publication in any other publication of computer-based information systems must be obtained from the Editor.

IBM believes the statements contained herein are accurate as of the date of publication of this document. However, IBM hereby disclaims all warranties either expressed or implied, including without limitation any implied warranty of merchantability or fitness for a particular purpose. In no event will IBM be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damage arising out of the use or inability to use any information provided through this publication even if IBM has been advised of the possibility of such damages, or for any claim by any other party.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

This publication may contain technical inaccuracies or typographical errors. Also, illustrations contained here may show prototype equipment. Your system configuration may differ slightly.

This publication may contain articles by non-IBM authors. These articles represent the views of their authors. IBM does not endorse any non-IBM products that may be mentioned. Questions should be directed to the authors.

This information is not intended to be an assertion of future action. IBM expressly reserves the right to change or withdraw current products that may or may not have the same characteristics or codes as those listed in this publication. Should IBM modify its products in a way that may affect the information contained in any user of the modification, it is possible, that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services. This is not to be construed to mean that IBM intends to announce such products, programming or services in your country.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. All specifications are subjects to change without notice.

With respect to advertising material herein, this publication does not constitute an expressed or implied recommendation or endorsement by IBM of any particular product, service, company, or technology.

IBM takes no responsibility whatsoever with regard to the selection, performance or use of the products advertised herein. All understanding, agreements, or warranties must take place directly between the software vendors and prospective users.

To correspond with the IBM OS/2 Developer, please write to the Editor at IBM Corp., Internal Zip 2230, P.O. Box 1328, Boca Raton, Fla. 33429-1328.

IBM, OS/2, Micro Channel, OS/400, Operating System/2, DB2, Application System/400, NetView, Systems Application Architecture, PS/2, AIX, PROFS, CommonView, CICS/400, and PS/2 are registered trademarks of IBM Corp.

SAA, CUA, Presentation Manager, IBM Proprinter, DRDA, Risc System/6000, SQL/DS, Common User Access, Print Manager, Workplace Shell, IMS, and CICS are trademarks of IBM Corp.

Windows, Microsoft, QuickHelp, and Network Driver Interface are registered trademarks of Microsoft Corp. Ellipse is a registered trademark of Cooperative Solutions Inc. Adobe Type Manager is a registered trademark of Adobe Inc. ActionMedia II and Audio Visual Connection are registered trademarks of Digital Video Interactive. HyperPilot and HyperACCESS/5 are registered trademarks of Hilgraeve Inc.

Netware is a trademark of Novell Inc. Novell and Btrieve are registered trademarks of Novell Corp. Touch Tone is a registered trademark of AT&T. dBase and Paradox are registered trademarks of Borland International Inc. Ethernet is a registered trademark of Macintosh is a trademark of Apple Computer Inc. UNIX is a registered trademark of UNIX System Laboratories Inc. MCI is a registered trademark of MCI. MCI MAIL is a trademark of MCI. VAX and DEC are registered trademarks of Digital Equipment Corp. HP is a registered trademark of Hewlett-Packard Corp. CompuServe is a registered trademark of CompuServe Inc. TYMNET is a trademark of B.T. North America.

Editor's Comments



What happened to the *Personal Systems Developer*? No doubt you have already noticed a few changes.

Last spring, the long-awaited OS/2™ 2.0 became generally available to our customers. This rebirth of the operating system coincided with many other changes at IBM, including an improved and expanded support program for our software developers. This magazine has been a key part of IBM's OS/2 support from the beginning. Now we're making a few changes of our own.

A New Look

Our new publisher is Miller Freeman Inc. in San Francisco, California, publisher of *Computer Language*, *UNIX Review*, *LAN Magazine*, and *AI Expert*. We are excited about this new partnership and the benefits it will bring to both the magazine and our readers.

A New Name

We have always been an all-OS/2 magazine. Now it's time to let our name reflect our true colors.

Tell Us More

We have been reaching OS/2 application developers through several channels: our own Developer Assistance Program, IBM's Mechanicsburg Distribution Center, individual subscriptions, and a variety of classes, seminars, forums, and trade shows. But we still don't know who *you* are. If you received a questionnaire with this issue, please take the time to fill it out and return it to us. We need to know who you are and how we're doing. What kind of work do you do? Which articles are most useful to you in your work? What new topics do you want to see? Do you have a

success story to share? Here's your chance to help us make this magazine the best it can be.

Welcome Aboard

Many of you are discovering us for the first time. That's because we are launching our retail sales campaign, taking *OS/2 Developer* to a variety of computer and book stores.

You've probably noticed another change: paid advertisements. We are opening *OS/2 Developer* to independent software vendors, OEMs, and other partners who have products of interest to OS/2 application developers. Software tools has always been a popular topic in these pages, and we expect the ads to announce new and useful tools, utilities, and other products to enhance your applications.

As one of my colleagues remarked recently, it's time for our "baby" to leave the nest and make its way in the world. As with OS/2 itself, we know we have a good product that can help thousands of software developers. To our long-time readers: thanks for your support and encouragement. To our new readers: we're glad to have you with us; don't be shy about telling us what you think.

There's one more change. Starting with the next issue, we're going to add a "Letters to the Editor" column. Here's a chance to comment on our articles, ask questions of our writers, and express your opinions about OS/2. Send your letters to me at MCI Mail (274-8797) or fax (407) 443-4233.

Dick Conklin
Editor



Dick Conklin

*A new OS/2;
a new magazine*



Spotlight

Spotlight on the OS/2 2.0 Development Team

by Mary A. Wright

In size, OS/2 2.0 is an awesome product, containing:

- 21 disks (33 MB) for the base operating system and its on-line help information and 400 pages of hardcopy documentation
- 9 disks (24 MB) for the toolkit (sample programs, development tools, and on-line programming information)
- 8000 pages of hardcopy programming information in the technical library.

In functionality, OS/2 2.0 is exciting, offering an integrating platform with an object-oriented user interface for DOS, Windows™, and OS/2 applications.

On March 31, 1992, OS/2 2.0 officially went golden. Sleepless nights of building driver after driver until all was right were over. Clusters of colored balloons, carrying invitations to celebrate the moment, glided down the halls of the Personal Systems Programming Center (PSPC) in Boca Raton, Florida. It was a moment of joy, relief, and great expectations. The OS/2 team had met the challenge and won.

The Spotlight article in each issue of this magazine usually features a key OS/2 software vendor and that vendor's experience with OS/2. In this issue, we depart from our usual format and spotlight the PSPC's OS/2 development team.

TEAM BACKGROUND

The OS/2 development team consists of men and women of diverse age, education, and experience. It's a fairly young

group of people. Most joined IBM directly after graduation from college. Most hold a bachelor's degree or equivalent in a scientific field. Some, like Greg Simco, technical lead for the OS/2 File System, are working toward doctoral degrees in computer science.

Although many team members have degrees in computer science, some have found their way into the world of the PC after academic work in other subjects. Bill Bodin, staff programmer and technical lead for OS/2 Console I/O, received his bachelor's degree in biology from the University of Georgia. After getting involved with PCs while working at the university engineering center as a biochemical lab technician, he became software technician for the center and was involved in the design, coding, and implementation of several real-time systems.



Scott Jones

Several OS/2 team members had previous experience in operating system development. Scott Jones, an advisory programmer for PM Graphics, worked on the Series/1 Realtime Programming System (RPS). Steve Glazer, staff

programmer and team lead for Functional Verification Test, worked on the Series/1 Event Driven Executive (EDX) operating system in communications.

Before working on the OS/2 operating system, many team members had no experience with a

project of such magnitude and complexity. Craig Bennett, senior associate programmer and team lead for Installation Services, worked on IBM hardware design tools. Terri Beck, staff programmer and former team lead for WIN-OS/2, came to IBM from Encore Computer Systems, where, she says, "the entire organization—planning, design, development, testing, documentation, packaging, customer support, etc.—was approximately the size of the OS/2 development organization. The work and problems were similar, but on a smaller scale."

Like many Floridians, the OS/2 team members come from across the country and the world. Not everyone on the team, however, resides in Florida. Programmers from other IBM locations came to the PSPC on temporary assignments. Programming talent was also imported from outside the U.S. Chris Andrew, David Kerr, Mike Perks, Jeff Moreland, Kelvin Lawrence, and Donald Hobern are all PM programmers from the U.K.

TEAM RESPONSIBILITIES

OS/2 team assignments covered the wide range of components that make up the operating system. Terri Beck was responsible for coordinating seamless Windows in OS/2. Scott Jones helped write the 32-bit graphics engine. Bill Bodin, the former biologist, was responsible for OS/2's virtual video drivers.



Phil Doragh

In many cases, team members worked on multiple components of the operating system. Phil Doragh, senior associate programmer for OS/2 File Systems, was the primary developer for the HPFS file system and a secondary developer for the

Boot Manager, the Installable File System Mechanism, the FAT and HPFS disk utilities, and System Boot. Craig Bennett worked on the design, prototype, and code for the system

installation program, as well as the Toolkit installation program. Steve Glazer worked on design and development of the Floating Point Emulator Support and on the verification tests for memory management, kernel services, and asynchronous communications. Greg Simco, the future Ph.D., worked on System Initialization and File Systems (FAT, IFS, and the CDROM file system).

TEAM SPIRIT

Successful completion of a project as large as OS/2 2.0 required a true team effort. The OS/2 team member does not work in a vacuum. A programmer who sits at his or her desk in solitude, coding and debugging, does not fit into this new world of software development. No one is simply a programmer. Everyone is touched by design, development, testing, and documentation activities within and outside his or her immediate development team.



Kip Harris

Because of the structure of the operating system's kernel components, Phil Doragh and his team spent much of their time working with testers. Kip Harris, staff programmer for MVDM and Device Drivers, estimates that at least half of his time was spent

working with other OS/2 development teams. "Designing the specifications and developing the code for a component is only part of a developer's responsibility," explains Kip. "Testing is also a substantial part of the job, as is assisting Information Development with technical publications."

Craig Bennett estimates that he spent as much as fifty percent of his time working with people from other development teams. "Because the changes for a particular build of the operating system or toolkit are brought together for the first time during installation, we have dependencies on all OS/2 teams. A large portion of the changes that are made to the system also require changes to installation:





laying out and placing system files on the user's hard disk, modifying an .INI file, or building a new CONFIG.SYS. Because the installation programs have a user interface, I also worked with the Usability, Common User Access (CUA), and Information Development teams. It was interesting to see how each group had a different view of how things should look and act."

TEAM COMMITMENT TO MARKET-DRIVEN QUALITY

Toward the goal that OS/2 be the best PC operating system and the operating system of choice, all team members are keenly aware of customer needs. Throughout the development cycle, the OS/2 team responded to feedback from internal, usability, and external test groups made up of a variety of users. Thirty thousand beta copies of OS/2 2.0 generated comments that helped shape the final product.

Steve Glazer notes that market-driven quality initiatives are reflected in every facet of OS/2. "We are driven to provide the customer with a product that will be a useful tool in developing great applications and that will add value to his or her products. With its ease of use, data



Steve Glazer

integrity, and robustness, OS/2 provides a vision of computing for the nineties."

Chris Andrew and the Workplace Shell™ team wanted the right solution for the new object-oriented user interface. "CUA 91 was the blueprint for the Workplace Shell," says Chris. "However, we felt some freedom to bend the rules defined by CUA and to work with them to reach the right solution. The Boca Usability folks and the test facilities that they have were very instructive in trying to see which solutions worked and, more importantly, which were confusing to the test participants. The usability folks

brought in many, many people to see if they could use OS/2, and our team was frequently in there watching them. We were usability-driven, so the Workplace Shell would be as usable as we could make it."

Because of substantial investments in non-IBM hardware, customers demand open systems. This means that for OS/2 2.0 to be successful, it must run on most or all major Intel 386-compatible workstations. According to Bill Bodin, "We took the attitude that if it doesn't work on OEM equipment, then OS/2 is broken." To meet this challenge, the OS/2



Bill Bodin

team, particularly those in the area of device support, found themselves working with hardware and software developers from other companies on a daily basis.

Bill Bodin, for example, worked closely with software engineers

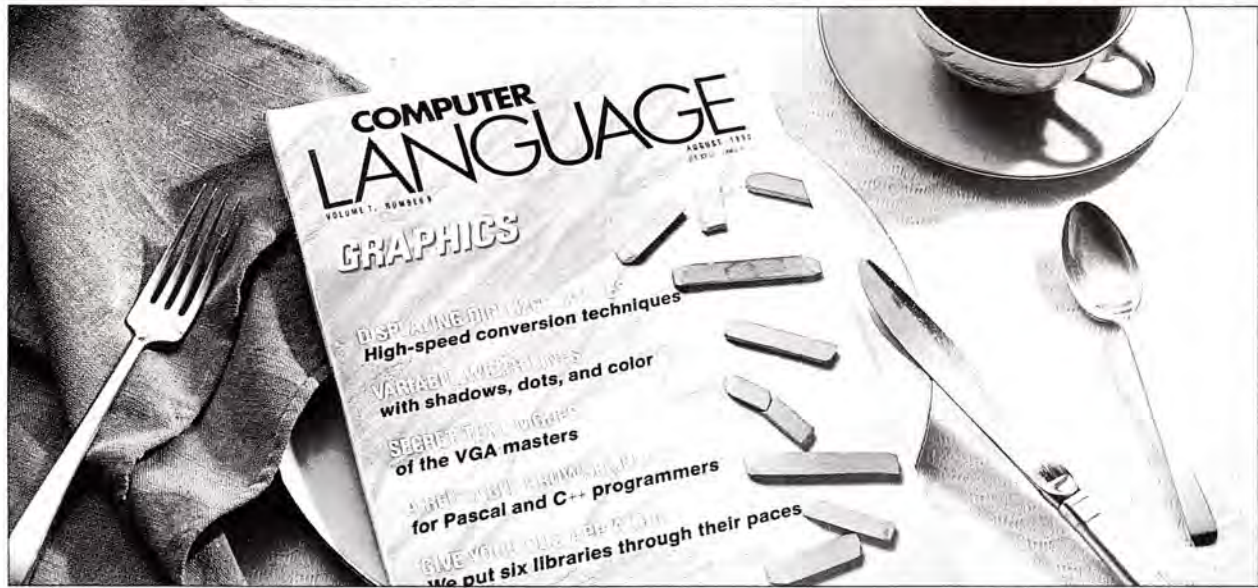
across the country to integrate code supporting a variety of OEM video adapters. Bill and his team assisted them with creating Presentation Manager™ display drivers. Kip Harris and his team shared a similar experience: "Many of these developers have joined us on site in Boca Raton, and many more have assisted us by telephone or fax," says Kip. "A lot of us are on the phone with technical people from another company at least once a week, and for some of us, like myself, it's perhaps a daily thing. This contact has been very beneficial, as developers from other companies give us considerable insight into the marketplace."

Even when meeting the challenges of supporting DOS and Windows applications, customer requirements were top priority. Terri Beck emphasizes, "Our entire design point was compatibility, compatibility, compatibility. We attempted to run every Windows 3.0 standard mode application off the shelf."

"If it doesn't work on OEM equipment, then OS/2 is broken."

—Bill Bodin

Stuff Yourself



All the programming you can stand for about \$2 an issue.*

Computer Language serves up serious programming, not just a lot of industry hype and copious amounts of C or Forth code for do-it-yourself toy programs.

You'll get object-oriented analysis, testing and quality assurance methods, techniques for enhancing reusability, effective maintenance strategies, and ways to bridge the gap between design and code. And it all comes from professional software developers who test their skills against programming challenges day in and day out.

But brace yourself for a deluge of information.

Computer Language is not for beginners. It is, however, the only magazine which truly prepares readers to meet the challenges of developing software in the 1990's.

So if you're a serious programmer whose appetite for information has never been satisfied, go ahead... stuff yourself with *Computer Language*!

Clip coupon and send to: *Computer Language*, P.O. Box 53525, Boulder, CO 80322-3525
or call: **1-800-288-1322**



51CL4



Yes!

I want to receive a full year (12 monthly issues) of *Computer Language* for just \$29.95, a 36%* savings off the single copy price. I understand that **I may cancel at any time and receive a full refund.**

Check one:

☐ Bill me

☐ Payment enclosed

☐ Charge to: ☐ VISA ☐ MASTERCARD ☐ AMERICAN EXPRESS

CARD #

EXP. DATE

SIGNATURE

NAME

ADDRESS

CITY

STATE

ZIP

* Savings are reflected off the single copy price of \$3.95. The basic subscription rate is \$34.95. Canadian/Mexican orders must be prepaid in U.S. funds with additional postage of \$6 per year. All other foreign orders must be prepaid in U.S. funds with additional postage of \$15 per year for surface mail and \$40 per year for air mail.



THE OS/2 TEAM ON OS/2 2.0

OS/2 team members view their product with pride and have great expectations for its reception in the PC community. Their views reflect a strong belief in OS/2's value and their deep commitment to market-driven quality.

Kip Harris believes that "among other features, the OS/2 2.0 DOS environment is a terrific development environment even if the developer is initially targeting only DOS platforms. DOS applications under development or test will not hang the system; multiple executions of the DOS program can be run and debugged simultaneously; and of course, multitasking capability permits a developer to edit source, compile, and debug an application simultaneously."

The ability of Windows and PM applications to communicate through dynamic data exchange (DDE) makes it easier to migrate gradually to PM. Terri Beck points out that "customers can replace pieces of their application set without changing their entire environment. They can use PM front ends to launch existing



Terri Beck

DOS or Windows programs so they don't have to migrate all their code to PM at once. They can also start migrating some of their Windows code to PM, placing communication links with the new PM code into the remaining Windows code."

Steve Glazer adds, "The flat design of user and system memory is a major innovation for using the full power of the 80386 memory management capabilities. OS/2 is a unique product in the marketplace; it adds value and immense capability to any personal computer configuration. It has everything a customer wants in an operating system. It works with a wide applications base, providing the capability to develop powerful 32-bit applications. It is reliable, providing

developers with a secure environment to create stable applications. And it is user-friendly."

Chris Andrew says, "With the object methods we've published, the programmer is able to get right in there and start adding his own object classes to the Workplace Shell.

Workplace objects benefit from all function defined for the workplace object class they are derived from. You don't have to worry about where your object is stored. You don't have to

write any code to support drag-drop or display an icon. This is all taken care of. All you need to do is sit down and write the object's new function. Also, by being object-oriented, your code will probably get a lot of side benefits like reuse and good modularity."



Chris Andrew

"Another benefit of Workplace," continues Chris, "is that all of the shell's features can be customized or removed by writing replacement object classes that either add or subtract function from the shell, whatever is required. For example, a corporate account may have strong views about its employees playing with the games and other fun stuff in OS/2 2.0. With just a little programming, that customer can disable everything except the authorized programs and procedures on the operator's machine."

Bill Bodin is particularly proud of the SuperVGA support now being introduced for OS/2 2.0: "This support gives PM display drivers written for many popular adapters the ability to set SuperVGA modes in protect mode. It also provides vast support for SuperVGA DOS applications that formerly didn't execute properly on OS/2. I am very pleased with the level of support we were able to attain for the general availability level of OS/2."

"Our entire design point was compatibility, compatibility, compatibility."

—Terri Beck

THE LIFE OF AN OS/2 DEVELOPMENT PROGRAMMER

Because of economics and a rapidly changing marketplace, IBM, like other companies, must do more with fewer people in a shorter period of time. Given the size and complexity of OS/2, this means that the lives of OS/2 development programmers are not easy.

Early in the product development cycle, their work revolves around coding, testing, and entering the public build of the product. In comparison to later phases of the development cycle, the programmers work at a slower pace, focusing primarily on code. This is not necessarily an easy task. For example, because of the newness of the Workplace Shell and the demands placed on the small Workplace Shell team for demonstrations, the team had to be physically removed from OS/2 development for approximately four months to complete their code.



Greg Simco

After the code enters the public build, the programmers' days revolve around problem determinations, fixes, testing the fixes, and making the next build. The demands on a programmer's time depend on the urgency or severity

of the problem, the difficulty of the problem's solution, and the frequency of public builds. Problems come from internal and external testers, from usability, and from anyone using a prerelease driver. Problems are classified by severity level based on the usability impact on the system or component and the availability of a workaround. A programmer's workload is prioritized according to the severity of problems reported against his or her component. Turnaround times for solutions are based on the severity of the problem, with more serious problems that impact the entire system requiring immediate attention.

As the development cycle for OS/2 2.0 progressed, the pressures on programmers, as

well as on the rest of the team, became enormous. Days became longer, with dinners served nightly in a patio area of the building. Long days turned into long nights and weekends. Weekly dinner menus grew to include weekend meals.

A project of such scope and importance requires serious commitment from all concerned. From the time OS/2 2.0 was launched in the spring of 1991 to its release on March 31, 1992, the OS/2 team worked large amounts of overtime—typically 60 to 70 hours per week. Some, like Bill Bodin and Greg Simco, worked 80 to 90 hours a week (twelve to fifteen hour days, seven days a week) in the last weeks of the development cycle. A sign on a Workplace Shell team office door, punning on the System Object Model (SOM) used to implement the shell, reflected the long hours of work.

The Workplace Shell Team
Suffers From
In-SOM-nia!

Some developers tried to reach a balance between work and home by creating individual work schedules with their managers. Kip Harris explains, "It wouldn't be unusual for me to take a few hours off midday to take my 3-year-old daughter to a library class. At another point, I might come in at nine at night and work several hours. Sometimes I would handle design work or correspondence from home after putting the kids to bed." Greg Simco, who normally goes to the gym six days a week, went to the gym whenever he could get away, morning, afternoon, or evening.

All members of the OS/2 team sacrificed some part of their personal lives to finish the project on schedule. They were unable to give normal attention to marriages, families, friends, hobbies, or special activities. Some were unable to juggle schedules and to spend quality time with their children. Some had to



Twenty-four hours after Windows 3.1 was released, os/2 2.0 ran a Windows 3.1 application.



place the pursuit of higher education temporarily on hold. Home maintenance became a very low priority.

All OS/2 developers were and are committed to the project, one they really believe in. Kip Harris summed it up: "One thing I see, over and over, is a great deal of personal commitment from the development team. We believe in this product and want to see it win. The developers have given and are continuing to give everything they've got to make it happen." This was most recently demonstrated at COMDEX/Spring in Chicago where, 24 hours after Windows 3.1 was released, OS/2 2.0 ran a Windows 3.1 application.

"OS/2? I've got my best people working on it!"

—Shon Saliga

As for the future of OS/2, Terri Beck says, "OS/2 must be a success. It's a good product, long needed by the marketplace. OS/2 is giving the PC the legitimacy found today in much larger systems. We have to reeducate programmers in the marketplace. They are used to programming any way they want under DOS—no rules. Can you imagine what would have happened if people writing apps for VM had that philosophy? OS/2 requires that level of discipline of programmers."

IN RECOGNITION OF THE ENTIRE OS/2 TEAM

The OS/2 development team is a unique group, "sharp and very creative," according to Craig Bennett, working in a "very challenging environment," according to Kip Harris.

But OS/2 development is only one part of the entire OS/2 team. Beside every developer is a planner, a system architect, a system tester, a performance specialist, a usability expert, an information developer, and a technical support person. And a strong management team (in fact, the entire corporation) stood behind the entire team, making it happen.

This commitment from management was very important to the OS/2 2.0 development effort. Bill Bodin says, "You could see the commitment in everyone from first-line managers to top executives. I remember on several occasions taking the long walk back from the OEM test lab well past midnight, and

Lee Reiswig (assistant general manager of programming for IBM Personal Systems) was still in problem tracking (PTR) meetings and still going strong. It is this level of involvement that kept us motivated despite the grueling schedule."

Another management team member involved in PTR meetings was Shon Saliga, OS/2 Systems Development Manager. Saliga tells us that his favorite quote from those meetings was "I've got my best people working on it." Says Saliga, "The dedication to making OS/2 a versatile product that DOS, Windows, and OS/2 customers would love was amazing. Even though great personal sacrifices were made, I believe people will look back and agree that this was a high point in their careers."



Tommy Steele

Former PSPC Director Tommy Steele, commenting on OS/2 2.0 and the OS/2 team says, "The development of OS/2 2.0 was truly a team effort. The men and women who developed this project took ownership and pride in their work

and made quality their highest priority. It was the first time that we opened up our development process to the public and made our people widely available to talk with customers and with the press. I believe the OS/2 team learned a lot about who our customers are, what they want in our product, and the importance of quality because of this new process."

"OS/2 2.0 really is a product that was owned by the people, not by the management team," adds Steele. "Working with our customers, the team determined what functions should be in the product and when the product was ready to ship. This customer-driven development process enabled them to build a quality product with superior technology that delivers the function that our customers want. I'm extremely proud to have been a part of this exceptional team."

If you're **not** interested in playing a *very* **COOL** 3-D

video game, spending a **MILLION** bucks, flying a



DC-10 into a **VOLCANO**, drawing on the wall



without getting in **TROUBLE**, watching **chemistry**

experiments **explode**, starring in your own **movie**,



helping black ants **fight** red ants, or learning how a



LITTLE box of wires and **CHIPS** can do all this, you **probably**

won't be **INTERESTED** in our **NEW** exhibit, either.

Get your hands on over 30 new and incredible ways
to use a computer at **The Computer Museum**
300 Congress Street, Boston. *Opens June 13, 1992.*
Call 617-423-6758 for more information.

Developed in collaboration with The Boston Computer Society.

Sponsors include 3Com Corporation, Apple Computer Inc., The Cabot Corporation Foundation, Digital Equipment Corporation, William H. Gates III, The Kapor Family Foundation, Arthur Nelson, Ingrid and Steve Stadler, Raytheon Company and Steve Wozniak.





Unfortunately, this article cannot cover the individual stories of each of the hundreds of people who worked on the project. However, we can share with you a team photo that includes many of those individuals. In addition, the entire OS/2 team is recognized within the installed OS/2 2.0 product. By clicking the mouse on the Desktop and then pressing SHFT-CTRL-ALT-o, you can view a bitmap image that scrolls the names of all of those who worked on the product.

Click on the
Desktop; press
Shift-CTRL-
ALT-o



Mary Wright, IBM Personal Systems LOB, 1000 N.W. 51st St., Boca Raton, Fla., 33429. Mrs. Wright is a staff information developer in Boca Raton. She came to IBM in 1982 as a developer of the Customer Support System, having previously worked as an applications programmer specializing in engineering applications. In 1985, she joined OS/2 development and was the original developer of the Monitor Dispatcher. In 1988, she joined information development, where she is the technical editor for the OS/2 Programming Library. She was the lead writer for the Application Design Guide and the three-volume Programming Guide in the OS/2 2.0 Technical Library. Mrs. Wright received a B.A. in Mathematics and Russian and an M.A. in Russian from Case Western Reserve University.

Windows

The All New Third Annual Summer

Windows & OS/2 Conference

65+ Tutorials, Conference Sessions & Fast Tracks ♦ Network Computing Showcase ♦ Multimedia Showcase ♦ Development Tools Showcase ♦ Free Test Drive Centers ♦ Hands-on Workshops ♦ Presentations Galore! ♦ 225 Top Exhibitors

August 19-21, 1992

(Tutorials: August 18)

World Trade Center, Boston

"Gold mine of ideas"

Lindsey Allen, Telecom Supervisor
Lockheed
Austin, TX

*"Excellent, no-hype presentations...
Very good common sense advice from
the trenches. An inspiration to all who
must compete today and beyond..."*

Stephen Richards, Developer
Fulcrum Technologies
Ottawa, Ontario, Canada

*"An absolute must for all Windows
users! It's hot!"*

Tony Odom
Systems Management Group
Hartford, CT

*"An innovative approach to solving
everyday Windows and Windows
application problems...and a well-
versed, informative presenter."*

Alan Gibbins, Mgr. Office Automation
Boeing Canada
Toronto, Ontario

FREE!

**VALUABLE INFORMATION ON
IMPLEMENTING WINDOWS &
OS/2--JUST FOR THE ASKING.**

We'll send you these valuable reports today, simply by requesting information on our next Windows & OS/2 Conference.

- ♦ **Windows 3.0 Character Set,**
from Brian Livingston
- ♦ **OS/2 2.0 Experts Advice,**
from San Francisco Canyon Co.
- ♦ **Windows 3.1 Tips & Tricks,**
from Mastering Computers



Windows & OS/2
CONFERENCE

Call 510-601-5000, Fax 510-601-5075 or Mail this coupon today for information on our upcoming Windows & OS/2 Conferences

Please rush me information on: ☐ Conference Program ☐ Exhibiting
☐ August 18-21, 1992, Boston ☐ January 19-22, 1993, San Jose

Mail me: ☐ Windows 3.0 Character Set ☐ OS/2 2.0 Experts Advice ☐ Windows 3.1 Tips & Tricks

Name _____ Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone _____ Fax _____

Please also send info to my colleagues at this location:

Name _____ Title _____

Name _____ Title _____

Phone 510-601-5000 Fax 510-601-5075

Windows & OS/2 Conference Produced & Managed by: CM Ventures, Inc. 5720 Hollis Street Emeryville, CA 94608

OS/2 is a registered trademark of IBM Corp.

Please attach your business card to this coupon

CL



Software Tools

Building Production-Quality Client/Server Applications



Jim Henry

by Jim Henry and Brent Williams

Since IBM introduced the IBM PC in 1981, transforming the microcomputer overnight from hobbyist's plaything into essential business tool, more and more business leaders have looked to integrate these machines into complex corporate computing networks. In time, companies made the machines an integral part of enterprise-wide network systems. However, despite these advances, the kinds of applications successfully implemented on networked systems haven't changed significantly. No matter how extensive the corporate networking vision, users are often willing to trust their networked microcomputers only with basic applications such as individual productivity packages, decision support, and electronic mail. Meanwhile, cost pressures continue to influence the movement of the organization's critical applications from mainframe systems to network-based applications.

The main barrier to adoption of client/server technology for production-quality applications is the concern that this environment cannot easily deliver the reliability of traditional mainframe computing. This concern arises primarily from software companies' lack of understanding as to just how much integrity and control is enough for true production applications. Another problem is the greater chance of failure in a client/server network. Instead of a relatively straightforward hardware path and one process to execute the application, a client/server application spans at least two machines and an enormous amount of system software: a user interface manager, a network manager, a database

manager, and so on. The odds are that a client/server network will be less reliable due to the large number of components that could fail.

Despite these problems, software technology has advanced significantly since the introduction of the first IBM PC. Because some programming environments (for example, OS/2) now support substantial amounts of memory and other capabilities, and because OS/2 and most database managers are now production-ready, it is possible to build programs that approach or exceed the requirements of production applications. With appropriate software unifying all components of the client/server environment and providing performance optimization and error recovery, production-quality applications can now be built on LAN-based systems.

Building robust client/server applications requires some of the most advanced features of today's operating and database systems. For example, OS/2 provides multitasking, interprocess communications, shared memory management, and other key features necessary to support sophisticated client/server functions. Relational database management systems (RDBMS) for OS/2 have evolved to the point that data integrity, high throughput, and network efficient versions are the norm. This article discusses the requirements that must be met for the client/server applications to be successful in production use. The support offered by development tools available for this environment will also be compared to the full requirements of production applications. Ellipse™, a product available from Cooperative Solutions, will be described to illustrate a new development approach that meets these requirements.



Brent Williams

REQUIREMENTS FROM THE DEVELOPER'S PERSPECTIVE

The key requirements of quality client/server production applications are similar to those of mainframe applications. However, in a client/server environment, additional capabilities must be built into the system to handle problems unique to client/server computing.

Integrity And Reliability

Users insist on the same standards of application reliability and data integrity for client/server applications as they do for mainframe applications. From a business perspective, a production system is critical; since lengthy downtimes to isolate software bugs or data corruption problems are unacceptable, the system users must be able to trust the system implicitly, without extensive human oversight.

Mainframe programmers create production-quality applications with the help of a number of system software components that support recovery from various error conditions. But these applications, such as IBM's IMSTM and DB2TM databases and the CICSTM teleprocessing monitor, are either not available or not sufficiently well accepted in the microcomputer community.

OS/2-based client/server development tools typically use only the limited recovery facilities in the database to recover from a failed in-progress transaction. Although the database typically rolls back the database component, programmers tend to rely on the RDBMS for all recovery functions. Unfortunately, the DBMS only insures that the database is returned to a consistent state after an aborted transaction, while network glitches and server faults can leave the client side of the application unstable. The client side of the application is thus left to the programmer to protect.

The client/server programmer is responsible for writing some reasonably complex code to first anticipate all recoverable errors and then restore the client to its exact state prior to the error. The programmer must protect working storage tables, variables, and the user interface.

Code must be written in C or in an arcane scripting language provided by the tool vendor.

In a client/server environment, transaction management actually becomes more difficult than in a mainframe environment. First, to deliver mainframelike capacity, client/server applications may be installed in dozens or hundreds of sites, pushing data closer to the customer. Or multiple machines may be employed to give higher performance on a single LAN. In either case, reliable transactions

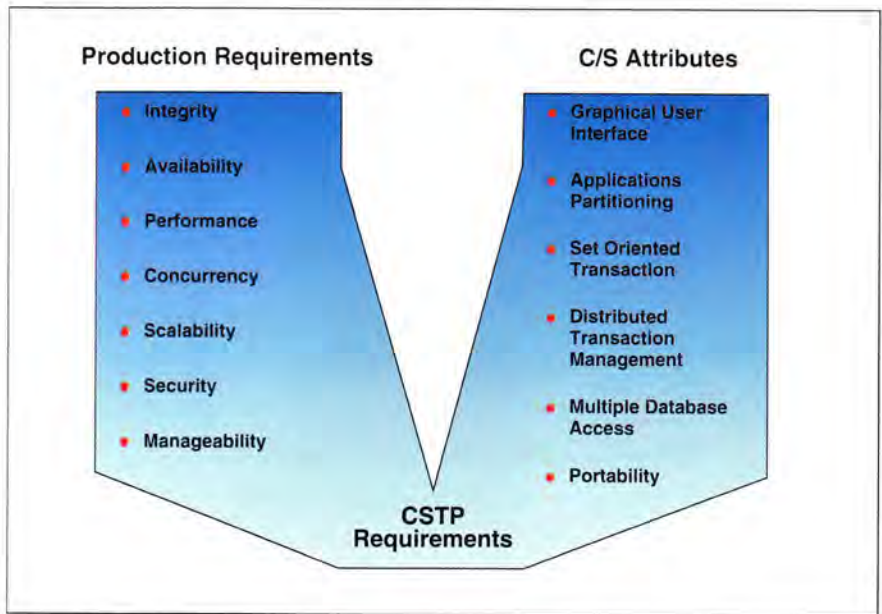


Figure 1: Requirements for client/server computing

will need to span more than one server machine. Although the technology for reliable updates, principally using the two-phase commit protocol, has been in existence for a long time, it has typically been difficult to implement, even with support from the database manager. What is needed to make real robust applications span the entire enterprise, then, is a transaction system that can provide reliable distributed updates with a minimum of programmer effort. Without this capability, users will turn away from distributed client/server applications.

Scalability

Scalability is the ability of an application or computing platform to grow to accommodate increasingly larger throughput. Client/server environments require more server power as



more clients are added. Server power can be expanded in three ways: you can replace small servers with larger ones, add processors to the multiprocessor server, or add more servers and redistribute the databases. As a result, it's important for client/server development environments to enhance the scalability of production quality applications without programmer difficulty.

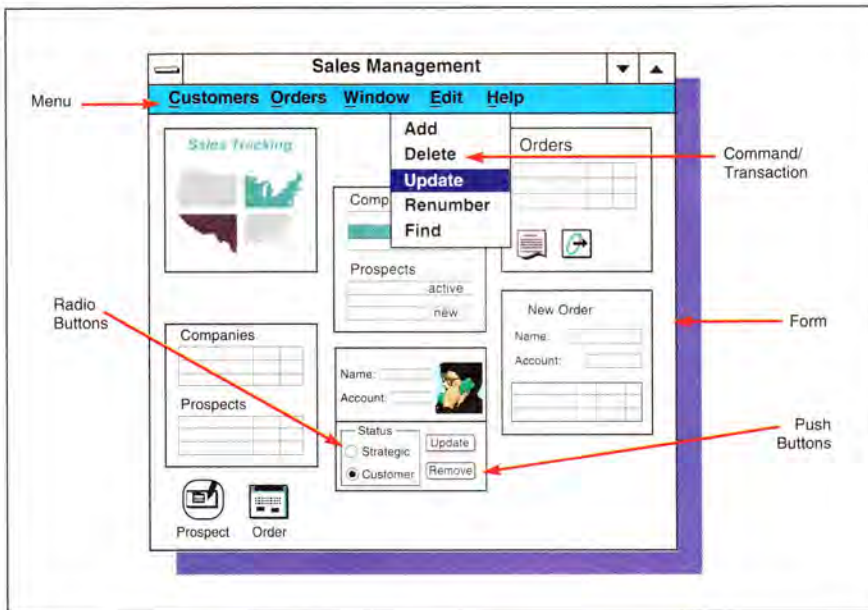


Figure 2: Graphic interface features supported in Ellipse

A client/server application demands integrity and control.

Performance

Production applications need to take full advantage of all the power offered by the clients and servers on a LAN. While there may be enough CPU cycles on the LAN to rival or exceed the available mainframe resources in a corporation, LAN computing power is distributed among a large number of small machines. Developers must therefore partition their client/server applications to run different parts of the application on the most appropriate machines. The primary goals of this distribution are the reduction of network traffic (a new problem for downsizing mainframe applications) and maximized use of available MIPS. In other words, if part of an application can run on the server platform containing the resource it uses (DBMs, communication, and so on), message traffic can be reduced and computation optimized.

Adding to the complexity is the fact that, in most organizations, the relative size of clients and servers can vary widely. Optimization, especially if required on a site-by-site basis, can represent a costly, repeated effort for many programmers.

Concurrency

Many client/server applications take advantage of today's powerful client workstations by selecting sets of records from the DBMS for local client-based processing. A scheduling system for a university might load records for all available classes to the client workstation as working tables. The user can then browse the working tables at speeds hundreds of times faster than could be achieved when directly accessing the disk-resident DBMS's tables.

Concurrency problems result if selected records are kept locked for the entire time they are used by one workstation. Most client/server databases operate this way, unless programmers enhance concurrency capabilities. The typical approach to solving concurrency problems is to try to reduce the amount of time a given record is locked. But if the records are not kept locked, updates can be made by other users, without the knowledge of the initial user. The result is hard-to-find data integrity problems. Developers who are aware of the problem have to resort to elaborate programming to prevent them, while programmers unaware of the risks can create applications that lose data and corrupt the database.

Mainframe programmers have used the "optimistic locking" paradigm, which allows the programmer to control the locking state of items in the database by writing substantial amounts of code. This method, however, typically requires substantial expertise to implement correctly.

It is important to understand what your underlying client/server database does, and more importantly, what you will have to do in your client/server development to get the required level of concurrent access despite the database's concurrency algorithms.

Security

Client/server application development tools depend upon the intrinsic security features provided by the LAN and DBMS. Production applications typically demand a higher level of security that limits users to specific applications, transactions within applications, and data items on screen displays. A human resources application might limit the users to human resources personnel and high-level executives, confine lower level employees to specific transactions that don't affect pay level, and suppress the display fields for salary amount for certain groups of users. The programmer must either custom code every application or write significant centralized run-time code to manage site specific security tables and to limit access to the protected objects.

Graphical User Interface

Because of the tremendous amount of low-cost computing power available on each user's desktop, client/server applications can use substantial computer power to make the user interface as easy to use as possible.

Programming this interface at a low level requires complex and hard-to-maintain code written in C. The "event loop" programming model has proven to be extremely difficult to learn and write reliable applications in. Most client/server development tools eliminate the need for this kind of hand-coding but do little beyond automating the user interface and client-side application logic.

Life Cycle Management

When critical applications are moved to a client/server environment, users need tools to manage complex projects. As client/server applications become more mainstream and the number of users increases, more client/server applications are being developed by teams of programmers who can modify and create modules without the awareness of the project manager or other programmers. The quantity of objects also continues to grow to an unmanageable number and chaos can result.

Client/server applications might also need to be deployed to multiple locations serving large

numbers of users. The resulting number of problems can make it extremely difficult to keep a project on track and nearly impossible to ensure that all workstations at all sites have current versions of code. Some add-on version control tools are available, but they cannot prevent a programmer from accidentally or deliberately circumventing their controls.

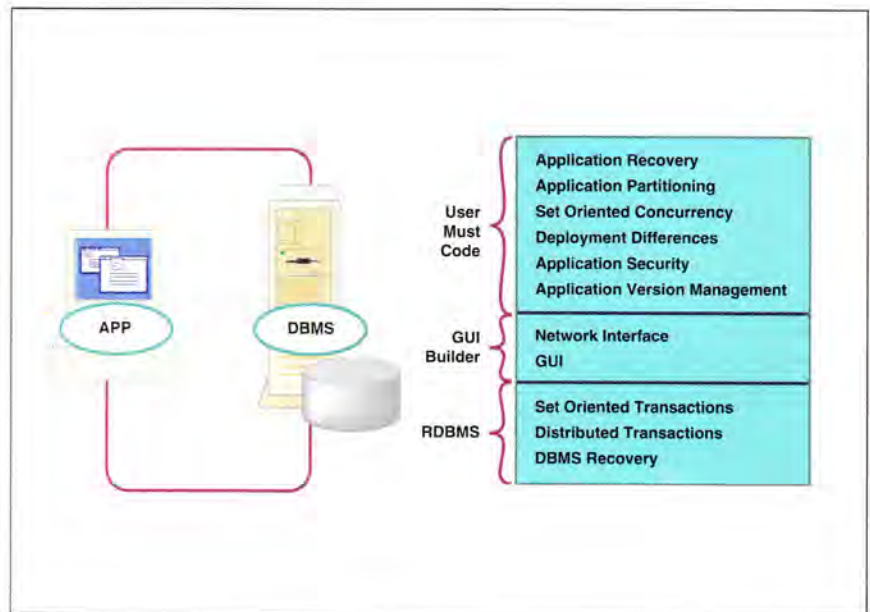


Figure 3: First generation client/server development

Replicated Applications

Corporations with many LANs often have standard applications that are centrally developed and then deployed to all sites. The client/server environment poses extra problems for the programmer trying to create an application that will run at all target locations, as different sites can vary as to DBMS vendor, location of database tables, printer locations, printer types, the client operating system, and so on. For example, the development and test site for an order-entry application might have all printers concentrated on one printer server and all DBMS tables on another server, while an installation site might have invoice printers attached to each client workstation and the DBMS tables split over two servers in two separate databases. The application must then be customized for each installation site.

PRODUCTION APPLICATION REQUIREMENTS VS. TODAY'S DEVELOPMENT TOOLS

The first generation of client/server development tools eliminated much of the coding previously necessary to create the graphical interface and simple database access parts of applications. Graphical user interface front-end tools obviate the need to use event-loop programming and hand-coded calls to graphical user interface toolbox functions.

Nonvisual functionality, however, still requires programming, as shown in Figure 3.

The main programming effort on client/server applications is spent writing database data manipulation language (DML) statements and utilizing C or a scripting language to meet the production application requirements just discussed.

ELLIPSE FOR PRODUCTION QUALITY CLIENT/SERVER APPLICATIONS

Ellipse, an integrated development and production environment from Cooperative Solutions, allows programmers to create production quality client/server applications without the need to resort to C programming. While the developer concentrates on the business functions of the application, Ellipse worries about error recovery and other production application requirements that would otherwise demand that code be hand written. The only coding the developer does is specifying the business logic for an application using a template-driven procedure editor. Ellipse consists of a development environment production system and a life cycle management system.

Ellipse Production System

The production system consists of several system software components that live on all participating machines in the network. The Ellipse/PS system software performs low-level system interfaces and recovery from system failures, precluding the need for the programmer to write system level code, as shown in Figure 5.

Multiple Ellipse server processes can be started to support symmetric multiprocessing. In addition, each server node has one monitor process, which provides process failure recovery and security management.

Features

To guarantee reliability and control in the client/server environment, the Ellipse/PS production system provides:

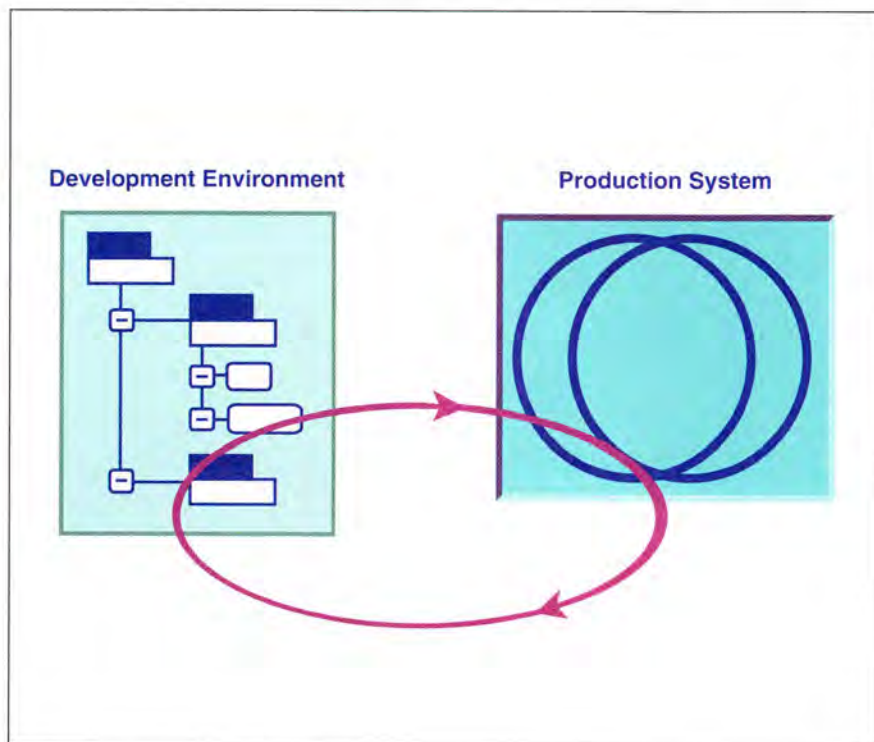


Figure 4: Application life cycle management

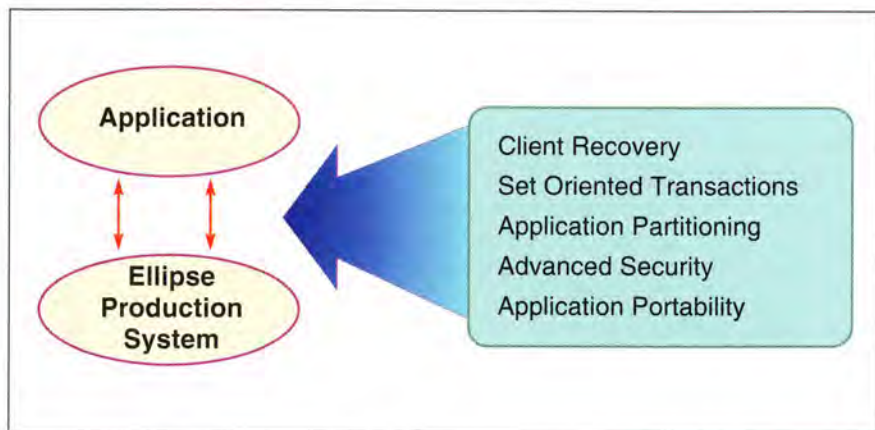


Figure 5: Ellipse/PS software layer automatically provides robust client/server applications

ENROLL Now!

DRAMATICALLY REDUCE YOUR DEVELOPMENT TIME

THE IBM DEVELOPER ASSISTANCE PROGRAM OS/2 2.0 (32-BIT) WORKSHOPS

Convert your existing DOS, Microsoft® Windows™ and OS/2® PM® applications to OS/2 with the assistance of OS/2 technical experts at a five day "hands on" workshop.

MOVE TO OS/2 2.0 – THE PLATFORM OF CHOICE

GET ON THE LEADING EDGE OF TODAY'S 32-BIT DEVELOPMENT ENVIRONMENT

RESERVE YOUR PLACE IN AN OS/2 WORKSHOP IN WEST PALM BEACH, FLORIDA

For Registration and Fee Information Call:

1-407-982-6408



October 12, 1992	■	DOS to OS/2 using VIO
August 31, 1992	■	OS/2 (16-Bit) PM® to OS/2 (32-Bit) PM
November 2, 1992	■	
September 25, 1992	■	OS/2 using VIO to OS/2 (32-Bit) PM
August 10, 1992	■	Windows 3.0 to OS/2 (32-Bit) PM
December 7, 1992	■	

ADDITIONAL CLASSES SCHEDULED QUARTERLY SPACE IS LIMITED!

RESERVE YOUR PLACE IN THE OS/2 DATABASE MANAGER PERFORMANCE WORKSHOP IN AUSTIN, TEXAS

For Registration and Fee Information Call:

1-512-823-2359

- July 21-24, 1992
- September 15-18, 1992
- October 13-16, 1992

© IBM Corporation 1992

® IBM, OS/2 and Presentation Manager are registered trademarks of International Business Machines Corporation
® Microsoft is a registered trademark of Microsoft Corporation
™ Windows is a trademark of Microsoft Corporation

- **Application Recovery.** All copies of Ellipse/PS cooperate for recovery from every detectable and recordable error a client/server application might encounter. Ellipse/PS manages and monitors the network connections, server processes, client processes, and each other so that errors are detected, and where possible recovered from, without any developer coding.

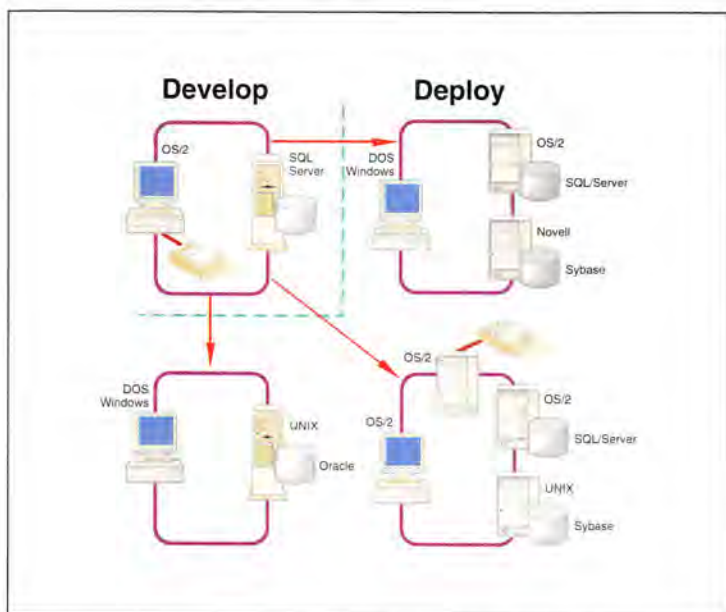


Figure 6: Applications may be built and then deployed to multiple configurations

- **Application Partitioning.** Ellipse/PS automatically partitions applications to best use the clients and servers while minimizing network traffic. The Ellipse Development Environment (DE) does not require the programmer to view the application as multiple processes, so programmers don't need to understand the client/server model or learn network APIs or RPCs. In unique circumstances, a system administrator can override automatic partitioning for site-specific reasons.
- **Efficient Set Concurrency.** The concurrency problem posed when a set of records is loaded into the client workstation for local processing is eliminated by the optimistic locking capability of Ellipse/PS, which will allow a user to manipulate records on the client without having to keep them locked on the DBMS. When the application attempts to update the DBMS with one or more changed record values, Ellipse/PS will

inform the application if one of those records was updated on the DBMS while the user was modifying the record in client memory.

Portability

Ellipse/PS always presents the same logical environment to the Ellipse applications, no matter what the physical environment. Ellipse applications are written once and deployed to any supported site configuration. At installation time, the application administrator maps logical resources (printers, DBMS tables, servers, and so on) to a site's physical resources. A simple visual utility is run to allow the administrator to create a list of resources that match the list of logical resources displayed by the Install utility. Applications are generated once and then deployed to all locations without any application changes, as shown in Figure 6.

ELLIPSE/PS AND APPLICATION LIFE CYCLE MANAGEMENT

Ellipse manages the life cycle of an application from the development of its first components (objects) through retirement. All objects (forms, applications, procedures, reports, configurations, and so on) are stored in the Ellipse repository. The Ellipse configuration management and version control system uses the repository to manage all phases of application life.

ELLIPSE DEVELOPMENT ENVIRONMENT

The multiuser Ellipse/DE incorporates all facilities needed to develop and deploy production quality client/server applications. Ellipse/DE runs on OS/2-based workstations and takes full advantage of OS/2 multitasking, interprocess communications, large memory model, and the power of the high-end workstations and servers for which it was designed.

With Ellipse/DE's integrated visual editors, a programmer without any experience in client/server technology can rapidly develop applications, without having to program in C or a scripting language. Even stored

procedures for the DBMS are generated by Ellipse and will automatically execute where they will be most efficient.

With Ellipse/PS taking responsibility for critical system-level functions, a developer can focus on the business functions being automated. CASE tools from a variety of vendors can be used to design a database to support the application. The application analyst then determines the business transactions that will manipulate the DBMS, generate reports, and interact with the users.

Each activity, a group of related business transactions, is implemented by specifying a set of forms, procedures, and reports, with the flow between them determined by commands from pull-down menus. In production use, commands are also chosen from the pull-down menus, which are created by the visual menu editor, as shown in Figure 7.

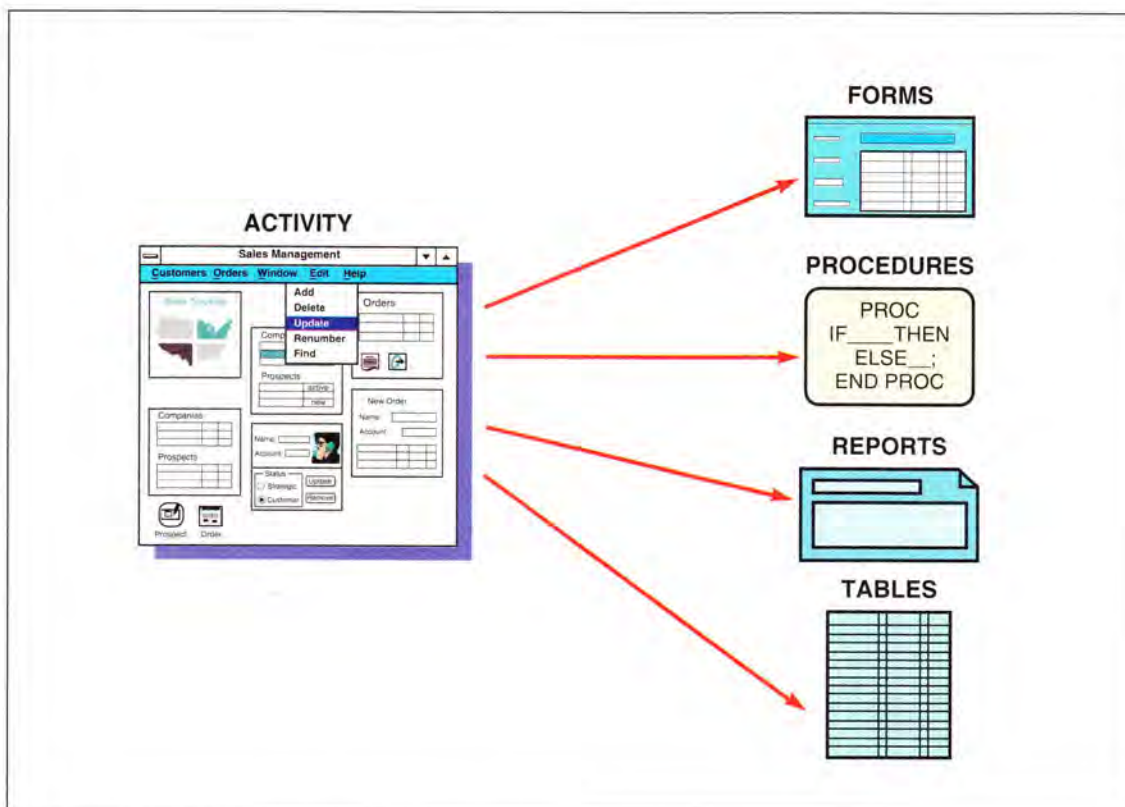
Commands can call procedures, bring up dialogue boxes, and run reports. Direct control given the user by the visual interface (menu bars, pull-down menus, radio buttons, and so on) eliminates the need to program

navigation logic. Menus and forms are created with their respective visual editors, without any programming necessary.

Procedures are easily created through the use of Ellipse's template-driven procedure editor. A programmer specifies procedure logic and data manipulation by indicating the statement type and then filling in blanks in the template (usually the variable names being tested or manipulated).

An application's overall structure is represented by a diagram of object relationships automatically created by Ellipse as the visual object editors create application objects, shown in Figure 8.

The development process is thus reduced to creating menus of commands, defining the commands with the command editor, and creating the forms and procedures using their respective visual editors. All testing, version control, configuration management, remote installation, and application management are achieved by Ellipse's integrated life cycle management.



Ellipse DE takes full advantage of the power of the high-end workstations and servers for which it was designed.

Figure 7: Integrated visual editors for rapid development



SUMMARY

Ellipse brings several functions to the OS/2 developer, among them rapid application development via visual editors, dynamic recovery from errors, integrated configuration management and version control, and parallel development and testing. Applications can be deployed without program change and are protected by advanced application, transaction, and field level security.

FOR MORE INFORMATION ABOUT ELLIPSE

For more information about Ellipse, contact Cooperative Solutions at 1-800-4-ELLIPSE or 1-408-377-0300.

Jim Henry is the Director of Technical Marketing for Cooperative Solutions Inc. He has over 22 years experience in the MIS field, with emphasis on OLTP, client/server processing, and DBMSs. He received a B.S. in computer engineering from Case Western Reserve University.

Brent Williams is a technical marketing manager at Cooperative Solutions. He is responsible for market analysis and creating the company's technical marketing publications. Williams has 15 years' experience in the software business, most of that as a developer of relational database engines and other system software. He joined Cooperative Solutions as its first applications engineer in 1990. He holds a B.A. in English literature from the University of California at Berkeley.

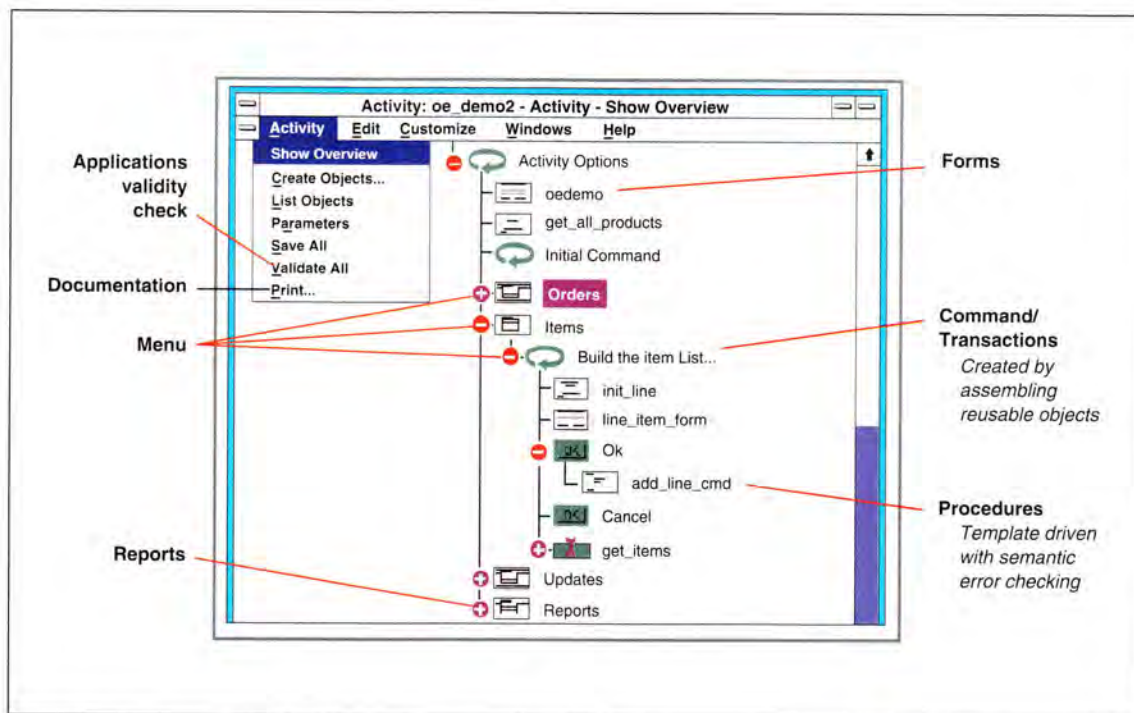


Figure 8: Ellipse's visual object editors and application overview diagram

“The Software Development conference plays an important role in helping developers keep their competitive edge... a must attend event.”



Ian McPhee
President
WATCOM Systems, Inc.

Follow the Leaders... ... to SD '92—Fall!

By now you've heard about Software Development Conferences. This year more than 5,000 of the leaders in the development field came to Software Development Spring — that's 100% more than last year!

Why were they there? To catch up on the latest trends in programming — from object-oriented programming in C++ to developing for Windows to managing the people aspects of the software development process.

They came to meet others who understand the significance of their efforts and achievements. They came to find answers to their current programming dilemmas. They came to rub elbows with the leaders in the software field, and to be part of an event that continues to shape the future of the software field.

It's not too late to take your turn at talking one-on-one with the best in the business. Software Development happens twice a year, and now's the time to plan to attend this fall.

At Software Development Fall the speaker lineup will represent the best and

the brightest — industry leaders like Gene Wang, P.J. Plauger, Charles Petzold, Steve Mellor, Tom Plum, Richard Hale Shaw, Bruce Eckel, Larry Constantine, Andrew Shulman, and Tom Swan to name just a few.

These aren't just great speakers, they are great teachers who can provide programming solutions you can take back to the office and put to use immediately.

Come to Software Development Fall and you'll leave full of new ideas, reenergized and ready to take on your next big challenge.

It all takes place the week of Sept. 14-18, 1992 at the World Trade Center in Boston. Mark your calendar now.



Drop this coupon in the mail today!

Or call us to be sure you're on our list. We'll rush you all the details on Software Development Fall. Learn about how to save money on advance registration. Plus, we'll keep you posted on up to the minute news about lectures, workshops and a wealth of vendor hosted events that will take place.

Call today at (415) 905-2741

YES!

Please send me more info on:

☐ ATTENDING

☐ EXHIBITING

Name

Title

Company

Address

Phone ()

Fax ()

Software Development '92—Fall
600 Harrison St. San Francisco, CA 94107

OSF



Software Tools

Application Development Aids for OS/2 2.0



Pamela Mitarotondo

by Pamela Mitarotondo

A number of application development products are available for OS/2 2.0. The IBM C Developer's WorkSet/2 is an entire programming environment that includes a C compiler, Presentation Manager debugger, configurable "shell" (the IBM WorkFrame/2), and the Developer's Toolkit for OS/2 2.0. The revised Toolkit has been improved but still includes the familiar necessities.

DEVELOPER'S TOOLKIT

The Toolkit contains the software tools needed to build and debug your OS/2 2.0 applications, sample programs to show you how to make use of the Application Programming Interface (API), and on-line information.

TOOLS

The Toolkit's array of tools is made up of build tools, Presentation Manager (PM) tools, libraries, and system debug support. Header files provide data types, data structures, and other OS/2 function definitions used during program compilation. C language header files have the .H extension. Figure 1 shows the Toolkit contents.

Build Tools (utilities used to build programs)

- **EXEHDR** enables a developer to display or change fields in the header of an .EXE file.
- **FWDSTAMP** is used to add entry points to a dynamic link library (DLL) file.

- **IMPLIB** creates an import library that resolves an application's external references to subroutines or procedures in DLLs.
- **LINK** is used to build 16-bit .EXE files.
- **LINK386** is used to build 32-bit .EXE files.
- **MARKEXE** is used to mark executable files as window-compatible and able to handle long file names.
- **MKMSGF** creates a binary message file from error, help, prompt, or general text information.
- **MSGBIND** binds a message file to an application's executable file.
- **NMAKE** automates the process of building applications.
- **PACK** compresses one or more files into a single file, optimizing disk space and I/O performance for application installation.

PM Tools (utilities used to build a PM interface)

- **Dialog Editor (DLGEDIT)** creates and modifies dialogue boxes.
- **Font Editor (FONTEDIT)** creates and modifies fonts.
- **Icon Editor (ICONEDIT)** creates and modifies icons and bit maps.
- **IPF Compiler (IPFC)** creates a binary image of on-line help information for applications or on-line documents.
- **Resource Compiler (RC)** creates a binary file containing PM resources (icons, menus, help

IBM offers a number of application development products for OS/2 2.0

tables, and string tables) that can be appended to an .EXE or DLL file.

- SOM Compiler (SC) creates header files, implementation templates, and a class definition file for the System Object Model (SOM) programming environment.
- Workplace Class List lists Workplace SOM classes and enables updating of the library.

Libraries (used as input to the Linker that resolves an application's external references)

- OS2286.LIB resolves application references to 16-bit OS/2 functions.
- OS2386.LIB resolves application references to 32-bit OS/2 functions.

System Debug Support

The system debug support environment is new in the Developer's Toolkit for OS/2 2.0 and includes the debug kernel, symbol files, a debug version of the Presentation Manager, and tools that support debugging efforts.

- MAPSYM is used to generate binary files that the debug kernel uses to associate a symbolic name with an address in memory.
- T is a terminal emulator used by the debug kernel to communicate with the system being debugged.

A full-function Presentation Manager debugger is available in IBM's C Set/2 Version 1.0.

SAMPLE PROGRAMS

Sample programs are working examples that demonstrate features of the operating system. Some of the familiar 16-bit samples have been rewritten for the 32-bit environment, and many new samples have been written to illustrate new functions, such as improved PM window controls and object creation and manipulation. Once installed, the sample programs appear in the Samples Folder and can be started from the Desktop or from the command prompt.

The sample programs can be classified as either Presentation Manager or non-

Presentation-Manager based programs. Samples in each category (**IMAGE** for PM and **WORMS** for non-PM) illustrate porting and migration to the new 32-bit environment.

Included in the PM samples is a template sample that demonstrates the structure common to all PM applications. Another sample implements the new PM window controls (Container, Notebook, Slider, Spin Button, and Value Set) and demonstrates how a PM application conforms to Common User Access requirements.

Other samples demonstrate window creation and manipulation, direct manipulation, drag and drop, retained graphics and transformation functions. A printer sample illustrates how to display and print text, graphics, metafiles, and bit maps.



Figure 1: Contents of the Developer's Toolkit

There are also object-oriented programming samples showing the SOM and Workplace programming environments. **ANIMALS** and **TP** are SOM samples showing subclassing, inheritance, polymorphism, and public and private methods. **WPCAR** demonstrates how to create a Workplace object.

In the non-PM category are memory management and dynamic linking samples as well as samples showing the use of threads, pipes, queues, extended attributes, and



semaphores. A set of REXX samples are also included to illustrate calling REXX from a PM application, use of the REXX subcommand handler and variable pool, and how a C application calls a REXX macro.

Getting Started, in the Developer's Toolkit for OS/2 2.0, includes a description of each sample program. The *Application Design Guide*, included in the OS/2 2.0 Technical Library and also available separately, contains a matrix showing the function calls used in each sample program.

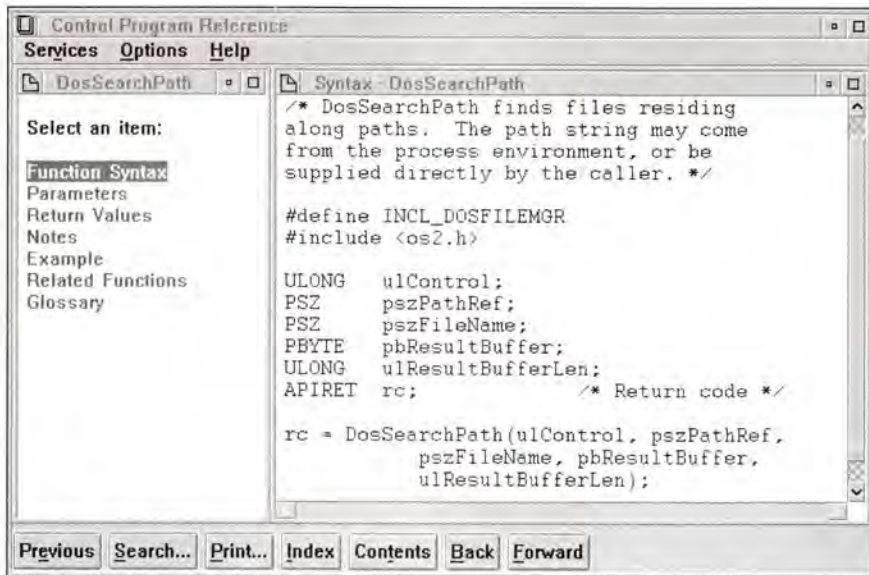


Figure 2: On-line information for the Developer's Toolkit

ON-LINE INFORMATION

The on-line information included with the Toolkit was created using OS/2's Information Presentation Facility and uses hypertext linking, searching, indexing, and cutting and pasting to the system clipboard. To better meet programmer needs, the Information Development team has redesigned the on-line information as a multiple-window presentation, as shown in Figure 1. Six documents cover the information needed to develop an OS/2 application:

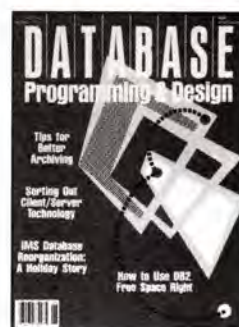
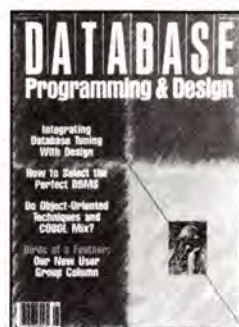
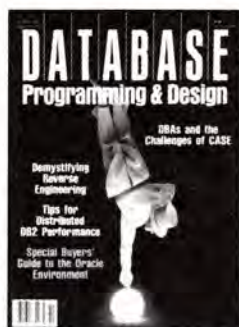
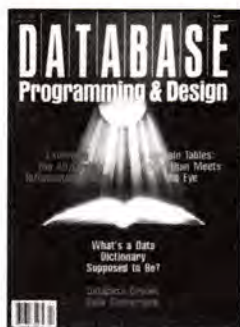
- The Control Program Reference describes each base operating-system function indicated by the *Dos* prefix.
- The Presentation Manager Reference describes each of the PM functions *Dev*

(device), *Drg* (drag and drop), *Ddf* (dynamic data format), *Gpi* (graphics), *Prf* (profile), *Sp1* (spooler), and *Win* (window). The PM Reference also describes graphics orders, application hooks, PM messages, and the new workplace methods.

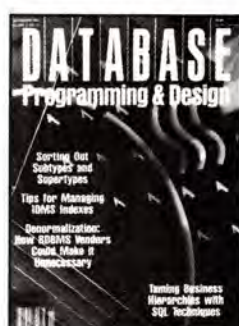
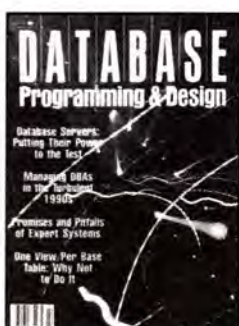
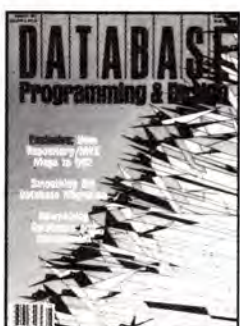
The Control Program and Presentation Manager references include input and output parameters, data structures, data types, return codes, and example code for their respective functions.

- The Information Presentation Facility Reference contains guidance and reference information for the design and development of on-line documents and application help facilities.
- The REXX Reference gives information on REXX functions, including call syntax, parameters, return values, and error messages.
- The System Object Model Reference provides information for each of the classes and methods in the SOM programming environment as well as SOM C language bindings, the Object Interface Definition Language syntax, and the SOM compiler command syntax.
- The Tools Reference describes each of the tools in the Toolkit, including system debug support.

KwikINF, a new Toolkit utility, allows hot-key access to on-line information, either from within an editor or from the command line. Once started, this utility sets off an automatic search for the text string of your choice. *KwikINF* is configurable, enabling you to specify the default hot-key sequence, default book to search, and whether or not to bypass the pop-up Search window; it also works in any of the OS/2 session types (full screen, AVIO window, and PM window). If your application is running in a full-screen session or in an active PM window with a multiline entry (MLE) field, *KwikINF* retrieves the word over the cursor and automatically places it in the entry field of the Search window. For non-MLE PM and AVIO windows, however, you must enter the string manually.



WE WOULD LIKE TO SURROUND YOU WITH SOLUTIONS.



Information at your fingertips.

It sounds like a cliché. But when you have a good magazine in your hands the cliché becomes a reality.

Every issue of Database Programming & Design puts information and solutions at your fingertips.

Solutions that will increase your database performance, reduce backlog and speed applications development.

Solutions that will, yes, make your job easier.

So no matter what your database specialty, you'll find a subscription to Database Programming & Design to be a real bargain. And at twenty percent off the regular subscription price, this bargain is now a real steal.

DATABASE Programming & Design



Solutions!

How can I possibly lose?
If I become dissatisfied with Database Programming & Design at any time, I may cancel my subscription and receive a full refund of my \$37.00 investment.

Check one:

- ☐ Payment enclosed.
- ☐ Bill me.

☐ Yes! Please start my full-year subscription to Database Programming & Design for \$37.00—a 21% savings.

Name _____ Title _____

Company _____

Address _____ City _____

State/Province _____ Zip/Postal Code _____

Note: Subscription rate is for US delivery only. Canadian delivery: US\$43.00. International delivery: US\$52.00 for surface mail; US\$77.00 for airmail. Please send payment with order in US funds drawn on a US bank. Please allow 6-8 weeks for delivery. CA residents add \$1.45 state tax.

3 Easy Ways To Subscribe!

☎ BY PHONE: 1-800-289-0169

☎ BY FAX: 1-415-905-2233

✉ BY MAIL:
Database Programming & Design
P.O. Box 53481
Boulder, CO 80322-3481



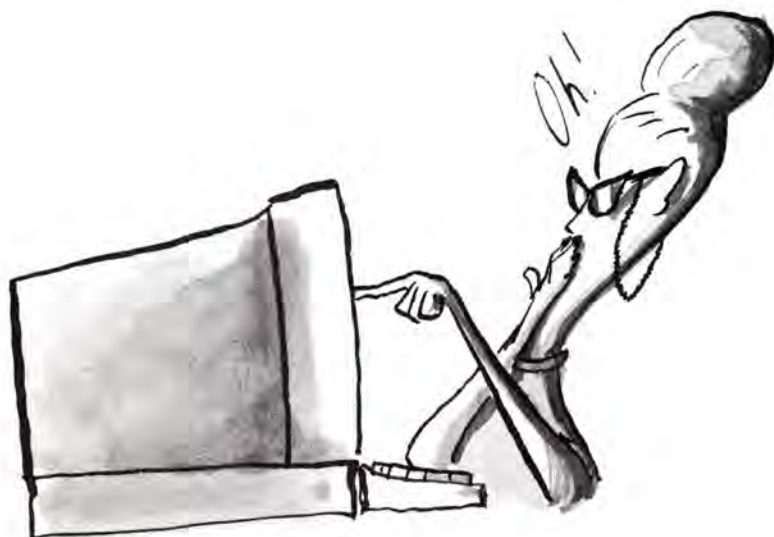
TECHNICAL LIBRARY

The OS/2 2.0 Technical Library has been revamped and improved as a result of customer feedback and competitive research. It includes familiar OS/2 programming information, with increased depth, breadth, and usability. The entire Technical Library is available in one package, or each book can be purchased individually as needed. An order form is included in the Toolkit package, or you can call 1-800-IBM-PCTB to order.

Each on-line Toolkit Reference (except the Tools Reference) is duplicated in the Technical Library, which contains the following volumes:

- *Control Program Programming Reference*
- *Presentation Manager Programming Reference, Volumes 1-3*
- *Information Presentation Facility Guide and Reference*
- *System Object Model Guide and Reference*
- *Procedures Language 2/REXX Programming Reference*

Additional books in the Technical Library include guidance for the OS/2 2.0 programming environment, device-driver information, and CUA information.



GUIDANCE LIBRARY

- The *Application Design Guide* provides an overview of OS/2 programming concepts, building dynamic link libraries, migration concepts, and information on using the System Object Model and Workplace programming environment. This book is a must for any OS/2 programmer.
- The *Programming Guide, Volume 1* provides in-depth information and code examples that will enable you to begin writing code using the Dos APIs. It includes sections on memory management, file systems and file management, program execution control, interprocess communication, exception management, and other base operating-system functions.
- The *Programming Guide, Volume 2* contains code examples describing the PM windowing functions. It includes information about window management, window controls, menus, dialogue windows, hooks, dynamic data exchange, direct manipulation, and other windowing functions.
- The *Programming Guide, Volume 3* consists of in-depth guidance and code examples describing graphics, printing, and device support in PM. It covers presentation space, graphics primitives, bit maps, retained graphics, metafiles, transformations, and other graphics functions. It also includes information on the print subsystem and spooling.
- The *Procedures Language 2/REXX User's Guide* describes the REXX language, covering basic and advanced topics for the REXX programmer. It includes descriptions of variables, expressions, commands, arithmetic, and other REXX functions.

DEVICE DRIVER LIBRARY

- The *Physical Device Driver Reference* covers the architecture and operation of physical device drivers. It includes information about character device monitors, OS/2 physical device drivers, DevHlp routines, and I/O control (Ioctl) functions.

- The *Virtual Device Driver Reference* contains information needed to write virtual device drivers, including virtual DevHlp functions and a description of each OS/2 virtual device driver.
- The *Presentation Driver Reference* describes OS/2 presentation drivers. It explains the internal interfaces between PM and the driver and between the driver and the I/O subsystem.

SYSTEMS APPLICATION ARCHITECTURE

- The *Systems Application Architecture: Common User Access Guide to User Interface Design* explains the principles, components, and techniques of user interface design and how to design a product with a CUA interface.

- The *Systems Application Architecture: Common User Access Advanced Interface Design Reference* lists all fundamental and recommended guidelines for designing and developing a product with a CUA interface.

ACKNOWLEDGMENTS

The following members of the Software Information Development department contributed to this article: Marion Bucko, Marion Lindsey, and Mary Wright.

Pam Mitarotondo, IBM Personal Systems Line Of Business, 1000 N.W. 51st St., Boca Raton, Fla. 33429. Mitarotondo manages the Software Information Development department responsible for producing Toolkit and Technical Library information. A 10-year veteran of IBM, she has held positions in Information Development and Programming.



Announcing TLIB™ 5.0! Version Control for DOS & OS/2

• The experts loved TLIB 4:

"...amazingly fast... TLIB is a great system." **PC Tech Journal**
 "TLIB has features and power to spare... TLIB is easy to use and the fastest of the reviewed packages." **Computer Language**
 "I will not program without it." **Uptime Magazine**

• Now TLIB 5.0 adds:

Automatic branching. Automatic version labeling across branches. Language-independent preprocessor, for "conditional compilation" in languages which do not support it (like dBase). N-way-tree version numbers. Branch and whole-library locking. OS/2 support.

• Plus the features they loved in TLIB 4:

Check-in/out locking. Branching, for parallel development. Keywords. Full binary file support (does not depend upon CRs in the file like other products). Wildcard and list-of-file support; can create lists by scanning source code for includes. Can merge (reconcile) multiple simultaneous changes and undo intermediate revisions. Network and WORM optical disk support. Mainframe-compatible delta generator for Pansophic, ADR, IBM, Sperry formats. Includes integrated PD MAKE by L. Dyer; also integrates with Opus™ MAKE, Slick™ MAKE, others.

MS-DOS \$139, OS/2 (with MS-DOS) \$195 + shipping.
 5 station network: MS-DOS \$419, OS/2 \$595. Call for other sizes.



Burton Systems Software

PO Box 4156, Cary, NC 27519 (919) 233-8128

World's most powerful tools for OS/2®

Hamilton C shell™

The superior alternative to the standard OS/2 command processor. Faithfully recreates and updates the entire UNIX® C shell language. Created explicitly for OS/2 using modern incremental compiler technology. Not one line ported from or created on anything but OS/2. Very high performance. Extensive support for multi-threading.

Features: Full-screen command line editing • Filename and command completion • History • Arrow and function keys • Unlimited size command lines • Recursive filename wildcarding • Fully nestable control structures • Command substitution • Aliases and shell procedures • PATH hashing • Background threads and processes.

Over 130 commands: alias, cat, chmod, cls, cp, cut, diff, dirs, dskread, dskwrite, du, eval, fgrep, grep, hashstat, head, history, label, ls, kill, markexe, more, mv, popd, printf, ps, pushd, pwd, rm, sed, sleep, split, strings, tabs, tail, tar, tee, time, touch, tr, uniq, vol, wait, wc, whereis, xd and others.

Meticulously adheres to OS/2 conventions. Supports HPFS, long filenames and all networks under OS/2 1.2 or later and 32-bit and VDM applications under OS/2 2.0.

\$350.00. Unconditional satisfaction guarantee. (\$365 in Canada, \$395 elsewhere.)

Hamilton Laboratories

13 Old Farm Road, Wayland, MA 01778-3117
 Phone 508-358-5715 • FAX 508-358-1113



Software Tools

The Enhanced Editor



Brian Proffit

by Brian Proffit

No additional charge." When's the last time you heard those words from an IBMer? On the other hand, how many times have you heard or read that there are no development tools available for OS/2? What's the likelihood, then, of there being a good development tool available from IBM at no additional charge?

Okay, okay, this column has said before that editors are not something you bring up in polite conversation. There are developers out there who are so devoted (even fanatical?) about their favorite editor that no other product has a chance of being considered. Such is life.

"No additional charge."

For those willing to look at other products, though, the new Enhanced Editor (EPM) for OS/2 2.0 is well worth a review. One of the strongest benefits, of course, is the price. It's exactly nothing, as EPM is included with OS/2 2.0. When you install OS/2 2.0, the Enhanced Editor is offered as an option under the "Tools and Games" selection. It takes nearly 900K of disk space, and end users probably won't install it (although perhaps they should). Every developer, however, should strongly consider doing so.

Let's establish one thing up front. EPM is *not* an "EDLIN replacement." That's what the system editor is for. EPM represents the fifth generation of an editor from IBM's Thomas J. Watson Research Center in Yorktown Heights, NY. These editors have been used inside IBM for a number of years, and that experience shows.

There are many features you would expect from a developer's editor. For instance, EPM

includes syntax assistance. If the extension of a file is .C, EPM assumes that the file is a C program and offers appropriate assistance. If the user types `if` and presses the space bar, EPM fills in the conditional block:

```
if () {
} else {
} /* endif */
```

Unlike many editors, however, EPM provides assistance for more than C programs. Help is available for Pascal and REXX programs as well. The same `if` statement for a Pascal program (extension .PAS) expands as:

```
if      then begin
end     else begin
end;    {endif}
```

The `if` statement for a REXX program (extension .CMD) expands into:

```
if      then
else
```

Other popular features such as cut and paste are also supported. Of course, since EPM is a PM program, its cut and paste isn't restricted to its own windows. EPM uses the OS/2 Clipboard to copy to and from other OS/2 programs such as spreadsheets, word processors, and host sessions.

Marking can be done in three ways. In line mode, complete lines are marked. In character mode, marking is linear. For example, if you start a character mode mark with the word "linear" in the previous sentence and end it here, the mark would follow around the line breaks to include only the words between the two marks. In block mode, the marked area is a rectangular region not necessarily related to



any word, sentence, or line boundary. Areas marked in block mode can then be copied, cut, moved, printed, and so on, regardless of geometry.

As a PM editor, EPM allows the user to take advantage of graphical user interface features such as the Adobe Type Manager, and supports all installed fonts. It supports multiple colors and attributes such as bold, italic, underline, and strikethrough.

Developers seldom request these features in an editor, thinking of them more as word-processing functions. But a developer can greatly increase productivity through their judicious use. For example, when you are paging through a program on the screen, having procedure and function declarations in a different color can quickly draw the eye to important parts of the program. Function calls, for example, could be done in a highlight color and comments in italics, making it much easier for a developer to locate the desired section of the program. Developers with an attitude will no doubt discover that comments they're forced to add to code can be done in fonts so small as to be unreadable:

```
main()           // My boss insists I write these dumb
  int stuff;      // comments all over the place, but that
  char *text;     // doesn't mean I have to like it.
```

Another way to move quickly to key sections of a program is through the use of bookmarks. The user can place bookmarks as needed through a file and assign them names to make them easier to negotiate in the future. Those with 400 lines of function code before the main program starts will appreciate the ability to move quickly to a bookmark at the beginning of main.

One of the best features of EPM is its support of macros, which are written in REXX. It would be difficult to imagine a richer language available to a programmer's editor. Through a simple interface, REXX programs can parse text directly from the editor, manipulate it, and pass it back. This fits directly into IBM's positioning of REXX as the SAA Procedures Language. I encourage all application developers to consider using REXX as the extension language for their product.

There are many other features that EPM has to offer, such as interfacing with the WorkFrame, drag-and-drop with the Workplace Shell, math functions, and draw mode. While I can't detail all of EPM's features here, I hope I have planted enough seeds of interest to encourage developers to study the online help included with the enhanced editor. This may be the greatest editor you didn't pay for.

Happy developing!

Brian Proffit, IBM Entry Systems Division, 1000 NW 51st St., Boca Raton Fla., 33429. Proffit is a senior programmer in OS/2 software developer strategy. He is currently responsible for OS/2 developer tools. He joined the IBM development laboratory in Boca Raton in 1983, after working as a systems engineer with IBM Dallas.





Software Tools

IBM Snapdump/2: Capture File for Problem Determination



Michael Garrison

by Michael Garrison and Cecil W. Rogers, Sr.

IBM SnapDump/2™ is a set of tools that assists with problem determination for OS/2 systems, capturing a wide variety of system data into a single, easily transportable file. The collection of data is controlled by a tailorable flat ASCII file. Although a sample version of this file is shipped with SnapDump (SNAPDUMP.DAT), the file can be easily modified to customize data collection. In addition, SnapDump provides an easy-to-use menu interface for displaying collected system data.

Data Collection

SnapDump includes a program (SNAPDUMP.EXE) that allows the collection of several types of data:

- Files (binary and ASCII)
- Data areas contained in named shared segments
- Standard output and standard error from programs invoked by SnapDump
- Environmental information automatically collected by the SnapDump data collection program.
- A system trace buffer (if the system trace is active).

Formatting Data

SnapDump includes a program (SNAPPDF.EXE) that provides a PM interface for viewing and formatting collected data.

SNAPPDF.EXE provides three views of the data that can be selected by toggling the PF12 key:

- Hexadecimal plus ASCII
- Hexadecimal plus EBCDIC
- ASCII only.

You can use the SnapDump formatter to extract data from the output file, returning it to its original binary format. This is particularly useful for binary files that can only be viewed using specialized tools or programs (for example, the Communications Manager configuration file). You can also extract executable files and use them to reproduce the problem on a support system.

Operating Environment

There are no special requirements for SnapDump other than the use of OS/2. SnapDump will operate in the following environments:

- SE 1.2 through OS/2 2.0
- EE 1.2 through EE 1.30.2
- SE 1.30.1 (including Extended Services)
- LAN Server 2.0

HOW DOES SNAPDUMP WORK?

SnapDump consists of a data collector and format utility.

IBM SnapDump/2 Data Collector

The SnapDump data collection program (SNAPDUMP.EXE) utilizes a flat ASCII file as input. This file contains the instructions that specify what data is to be collected (for



Cecil W. Rogers, Sr.



```
* OS/2 information
f/config.*
f/syslevel.*
f/startup.cmd
f/*.ini
f/os20001.dat
f/log0001.dat
*
* FFST/2 files
*
f/epw.ini
f/snapdump.dat
f/os2mlog.dat
p/dir c:\os2\system\*.dmp
*
* Communication Manager
*
p/display.exe
f/*.cfg
f/*.ndf
f/*.sec
f/*.cf2
f/error.dat
f/message.dat
f/esinst.hst
f/acs.pro
*f/acslan.log
f/lantran.log
f/c2instal.log
f/install.log
f/message.log
f/custbld.hst
f/cmfeater.dat
m\sharemem\acs\ras_seg
*
* Database Manager
*
```

```
f/sqlbdir
f/sqlsystem
f/sqldbcon
f/sqluif.*
f/sqlnodir
f/sqlgwdir
f/sqllogctl.lfh
f/dbdrqlib.cfg
*
* LAN Server 2.0
*
f/net.err
f/net.acc
f/net.aud
*
* LAN status command
*
*p/net statistics requester
*p/net statistics server
*p/net config requester
*p/net config server
*p/net files
*p/net sessions
*p/net share
*p/net status
*p/net view
*p/net who
*
* Workstation Status Programs
*
*p/netsess2.exe
*p/ncbstat.exe *
*p/dirstat.exe
p/findseg -S C:\
p/qmc.exe -d
p/pstat.exe
```

Figure 1: SNAPDUMP.DAT file

example, files and memory areas) and which data gathering programs are to be invoked. The flat file approach makes data collection with SnapDump very flexible because it makes it easy to add or remove items to be collected. When run, SnapDump will produce a single output file that contains all collected data.

SnapDump is invoked from the command line through one of three forms of the command:

- **SNAPDUMP**—This command assumes that the input file, `snapdump.dat`, exists, and that it

contains a list of files, data, and programs to be processed. The output file defaults to `snapdump.dmp`.

- **SNAPDUMP {input-fname}**—The {input-fname} parameter is the name of the flat ASCII file containing the list of items to be captured. The output file defaults to `snapdump.dmp`.
- **SNAPDUMP {input-fname dump-fname}**—The {input-fname} parameter is the name of the flat ASCII file containing the list of items to be captured. The {dump-fname} parameter is



the file into which SnapDump will write all the captured data. In this document, it is also called the SnapDump output file.

The SnapDump Input File

The SnapDump input file controls the information to be captured in the output file. It may be edited with any ASCII editor, including the OS/2 system editor. The types of information that can be captured (and how that information is specified in the flat file) are listed below.

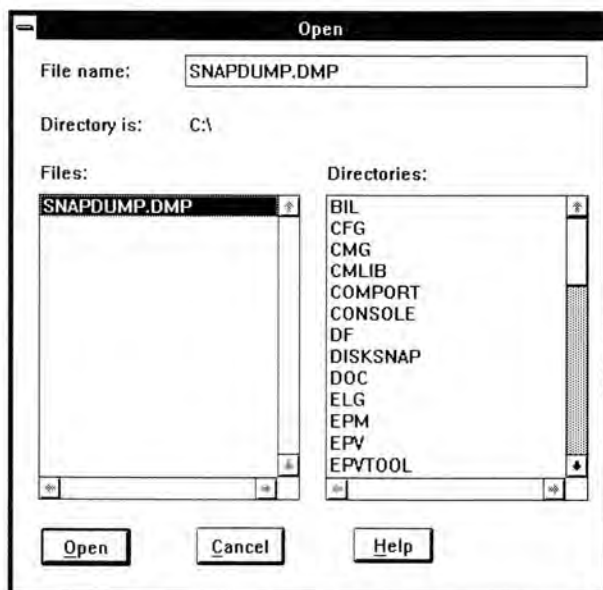


Figure 2: File-open dialogue

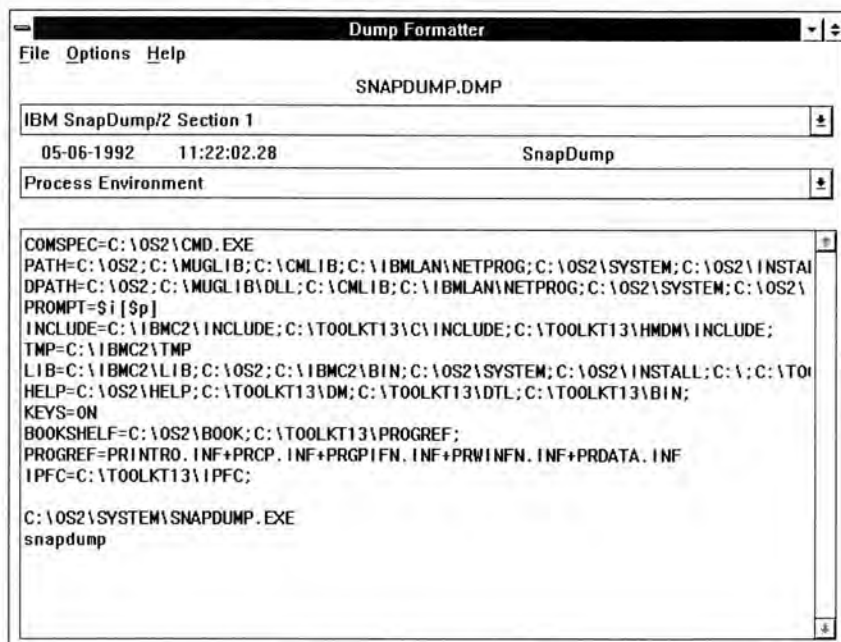


Figure 3: Maximized first window of SnapDump/2

- An **f/** in column 1 indicates that a file is to be captured, as in **f/config.sys**. When a file is captured, the contents of that file are appended to the SnapDump output file. Wildcards may be specified, but be aware that the SnapDump output file may become very large if the wildcard specification is too broad. Any files in use, such as the Communications Manager **MESSAGE.DAT** and **ERROR.DAT** files, are collected.
- An **p/** in column 1 indicates that the specified program is to be invoked, as in **p/c:\qmc.exe -d**. You can specify any programs (including .CMD files) that can be run from the OS/2 command prompt and write to standard output or error. The output of the program is appended to the SnapDump output file. Output from programs that use other techniques of writing to the screen, such as Presentation Manager **Win** or **VIOW** calls, cannot be captured by SnapDump.
- An **m/** in column 1 indicates that a named shared memory segment (**\SHAREMEM\...**) is to be captured, as in **m/\SHAREMEM\ACS\RAS_SEG**. The contents of the segment are appended to the SnapDump output file. Wildcards for named shared memory are not allowed, and certain segments are protected and cannot be dumped. You can obtain the names of the named shared memory files in the system by issuing the command **pstat /m** at the command prompt.
- Any other value in column 1 causes the line to be treated as a comment.

If the system trace is active, the system trace buffer is automatically captured by SnapDump and appended to the SnapDump output file.

Searching for Files

When SnapDump scans for files, it will search all accessible drives, including shared drives on servers and disk drives. If multiple instances of a file exist, all will be collected.

Sample SnapDump Input File

The input file **SNAPDUMP.DAT**, shown in Figure 1, is included with the SnapDump/2

package. It can be modified as required with your editor.

The SnapDump Format Utility:

The format utility provides a simple menu interface to select and display the contents of the SnapDump output file. Because each data item collected in the dump file is tagged with an entry title, it is easy to select a specific data item for display without displaying the entire dump file.

The format utility can also be used to extract a specified data item and return it to its original binary format. The file can then be extracted from the dump file and stored individually. (This is particularly useful for binary files that require specialized formatters.)

You can access the format utility directly from the command line by entering `SNAPDF {dump-fname}`. The `{dump-fname}` parameter is the name of the file containing data previously captured by SnapDump.

If the `{dump-fname}` parameter is supplied, or `snapdump.dmp` exists, the "Dump Formatter" window will be displayed.

If no dump file name is specified and a file named `snapdump.dmp` doesn't exist, the standard file-open dialogue box will appear to assist the user in selecting a dump file name, and the file-open dialogue will appear as noted in the **Open** window, as shown in Figure 2.

You can use either the mouse or the Tab and Shift keys to move between controls in the dialogue box. To move between the **Open**, **Cancel**, and **Help** buttons, use the left and right arrow keys. A user can press the F1 key or click on the **Help** button to obtain context-sensitive help on a selected control. The "Directory is:" field is read-only and allows the user to see long subdirectory names.

Only files identified as SnapDump or First Failure Support Technology/2 (FFST/2) will appear in the **Files:** listbox. The user may enter

It is easy to select a specific data item for display without displaying the entire dump file.



Gpif 2.0

Available Now
32 Bit Code Generation
CUA '91 Controls

Gpif

SYSTEMS, INC.

Let Gpif write the GUI you design

Using the powerful point and click visual programming environment of Gpif*, you can prototype, test and generate a complete OS/2 PM GUI in a few hours or days rather than the weeks or months required to hand code the same design. Even a relatively simple GUI can require writing thousands of lines of code, but with Gpif you simply draw your user interface on the screen. The integrated dialogue editor of Gpif permits actions and context sensitive help to be linked to controls as you create them. Gpif then generates error free ANSI C complete with embedded SQL statements.

Gpif is optimized to take full advantage of OS/2 PM, the most powerful and robust GUI system available. Since Gpif code directly accesses the PM API, there is no run time module to distribute with your application and no added overhead or royalties.

Gpif keeps the entire design definition in one file. This permits easy re-entry for updates as well as regeneration for different targets. 32 bit OS/2 and 16 bit OS/2 code generation.

Gpif supports:

- Simple and direct linkage of the interface to program logic, built in or user defined functions.
- Direct association of help screens with controls, and complete integration into the PM Help Presentation Facility.
- Flexible use of Presentation objects (fonts, colors, etc.) with controls and windows (client area and frame).
- Simple inclusion of bitmaps for use on About screens, user-defined buttons, and menu or pulldown entries.
- Automatic embedded SQL statements to read OS/2 DataBase Manager tables, directly into combo or list boxes.
- Multi-thread programming.
- Multiple source file generation.
- Automatic creation of controls that scale with window size.
- Inclusion of user defined controls

Try us out.

Order Gpif today for just \$995.⁹⁹

Call Gpif Systems Inc. at:

(203) 873-3300 or (800) 831-0017 - fax (203) 873-3302. Free demo software available

P.O. Box 414, 30 Falls Rd., Moodus, CT 06469

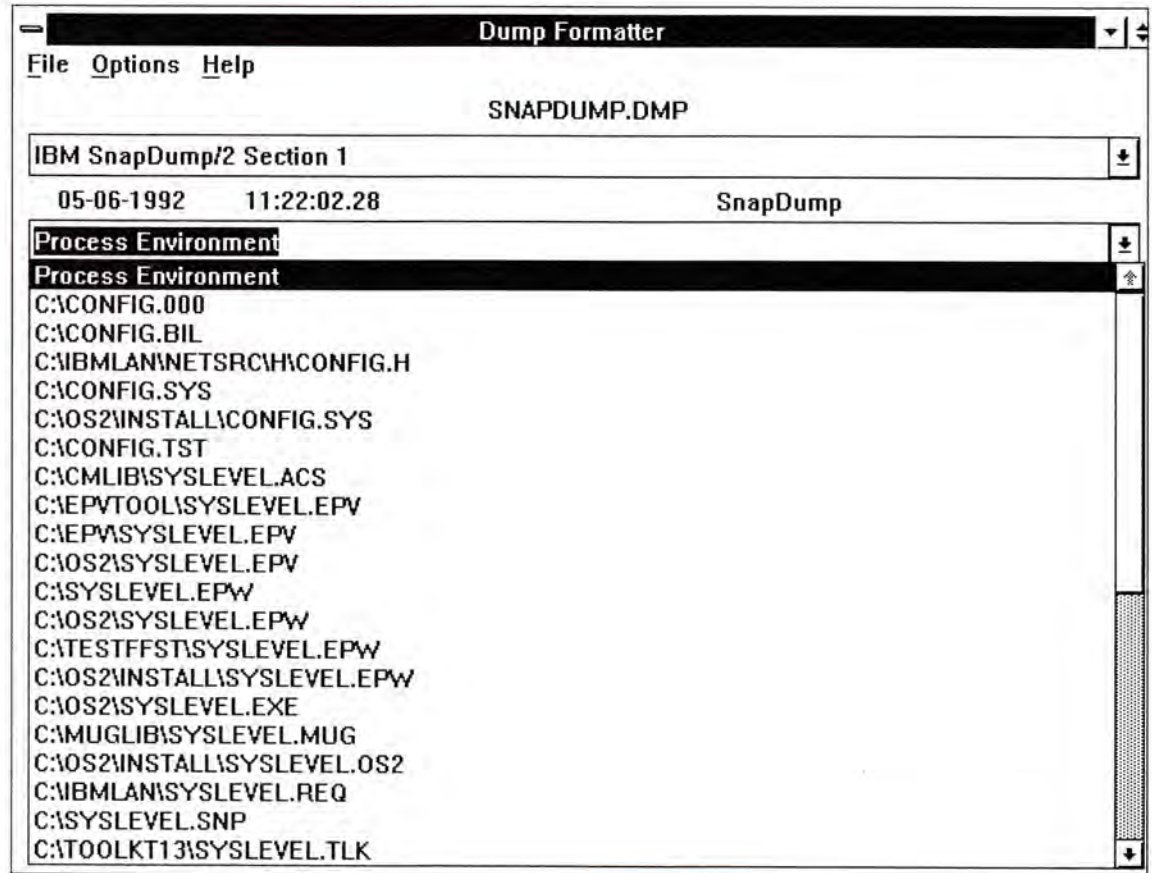


Figure 4: SnapDump data list

any name in the **File Name:** entry field; the dump formatter will assess whether the selected file is a SnapDump or FFST/2 dump file.

Once a dump file and the **Open** button are selected, the Dump Formatter window will display the formatted SnapDump data and a list of captured data. Figure 3 shows the Dump Formatter window.

The first field in the window names the dump file currently being displayed (here, SNAPDUMP.DMP).

The second field in the window identifies a dump segment. If the SnapDump file contains more than 300 data entries (such as capture files, output from programs, or shared segments), it will be segmented. Use the down arrow to the right of the field to see and select the desired segment.

The third field shows the date and time of the dump as well as the file's originating program (in this case, "SNAPDUMP").

The fourth field contains the selected data item. When the Dump Formatter window is first displayed, "Process Environment" is preselected. The environment information collected by SnapDump includes PATH;, DPATH;, and so on. To get a complete list of captured items, click the down arrow or press Enter. Figure 4 shows a data list.

When the Dump Formatter window is first displayed, the sixth field contains an ASCII representation of the Process Environment. The data can be displayed in one of three formats (use PF12 to toggle between data views):

- Hexadecimal plus ASCII (Figure 5)
- Hexadecimal plus EBCDIC
- ASCII only (Figure 3)

In Figure 4, a data item is selected for display by either clicking on the data item and pressing Enter or double clicking on the item.



Dump Formatter

File

Options

Help

SNAPDUMP.DMP

IBM SnapDump/2 Section 1

05-06-199211:22:02.28SnapDump

C:\OS2\EPW.INI

Offset	Hexadecimal				Translation
00000000	FFFFFFFF	14000000	15010000	00000000	
00000010	00000000	00000000	2C000000	00000000	
00000020	04000400	28000000	43464700	4B000000	 { ...CFG.K...
00000030	00000000	04000400	44000000	03000300	D.....
00000040	48000000	4D534700	4F4E006A	00000000	H...MSG.ON.j....
00000050	00000004	00040063	00000003	00030067	c.....g
00000060	00000058	53440031	35008900	00000000	...XSD.15.....
00000070	00000400	04008200	00000300	03008600	
00000080	00005841	44003132	00B40000	00000000	..XAD.12.....
00000090	00040004	00A10000	000F000F	00A50000	
000000A0	00534450	00433A5C	4F53325C	53595354	.SDP.C:\OS2\SYST
000000B0	454D5C00	DF000000	00000000	04000400	EM\.....
000000C0	CC000000	0F000F00	D0000000	41445000	ADP.
000000D0	433A5CAF	53325C53	59535445	4D5C0000	C:\OS2\SYSTEM\..
000000E0	00000000	00000004	000400F7	0000001A	
000000F0	001A00FB	0000004D	4C4E0043	3A5C4F53	MLN.C:\OS
00000100	325C5359	5354454D	5C4F5332	4D4C4F47	2\SYSTEM\OS2MLOG
00000110	2E444154	00			.DAT.

Figure 5: Dump formatter window in Hexadecimal plus ASCII format

The F4 key (or clicking on the down arrow) toggles the visibility of the list box of the prompted entry fields whenever the entry field has the keyboard focus. To select an item from the dump or entry lists, use the arrow keys or the mouse, and double-click on the entry field. When a new entry is selected, it will be formatted and displayed.

Items in quotes represent programs invoked by SnapDump. Output from these programs can be viewed as you would view a file.

Files captured in a SnapDump output file may be copied to disk with the **SAVE AS** command. In the **SAVE AS** dialogue box, set "Select output type" to binary data. Files such as OS2.INI, for example, should be saved as binary data to preserve their format. This function will not capture OS/2 Extended Attributes.

Multiple SnapDump files can be opened with the **Open** command.

Transporting SnapDump Output Files

If a SnapDump file is too large to fit on a single disk, you can use the **Backup** function of OS/2 to store it on multiple disks. Conversely, the **Restore** function can consolidate disk-spanning information into a single file.

PROBLEMS SOLVED WITH SNAPDUMP

A user with software problems often needs to capture problem-related information requested by support personnel. SnapDump does much of this work, locating and gathering diverse files.

SnapDump can perform Problem Determination/Problem Source Identifier work. For example, if a customer running the IBM ES Communications Manager for OS/2 changes the **STARTUP.CMD** to use a different configuration file and forgets he or she has



*SnapDump/2
can capture
problem
elements from a
remote location,
then analyze
them offline
while the remote
system
continues
processing.*

done so, it could cause the Communications Manager to run incorrectly. Forgetting the changed configuration files, the customer calls a support team, which requests eight files scattered across several directories. With SnapDump, the customer can easily collect all files for comparison, including the configuration file from the previous day.

In a second example, a SNAPDUMP.DAT file can be edited to capture specific files needed to resolve a problem (executable code, configuration files, and error data file) and sent to a user, who simply returns the collected files to his or her support team for review and repair.

Companies with internal support personnel will find SnapDump/2 valuable, as it can capture problem elements from a remote location, then analyze them offline while the remote system continues processing.

In the final example, a change to the CONFIG.SYS hangs the system during the IPL. To create a dump file in SnapDump/2:

- Execute IPL from the A: drive with the install disk
- Hit Enter to continue or ESC to cancel the operation
- With SnapDump/2 installed either on the C: drive or from a disk, execute SNAPDUMP.

SNAPDUMP.DMP will be captured and placed on a disk to be viewed from a fully initialized OS/2 system. After being identified, the problem may then be corrected and the system ReIPLed.

Michael Garrison, IBM Personal Systems Programming Center, 11400 Burnet Rd., Austin, Texas, 78758. Garrison is an advisory programmer for Extended Services for the OS/2 Communications Manager. He joined IBM in 1979 and initially worked on the 5520 Administrative System as well as serving as the development lead for the SNA Distribution Services (SNA/DS) feature of the 5520. In recent years, he has worked as a designer and developer for the IBM OS/2 Extended Edition and Extended Services Communications Manager. He received a B.S. in computer science in 1977 and a master's degree in computer science in 1979, both from the University of Southwestern Louisiana.

Cecil W. (Bill) Rogers, Sr. IBM Personal Systems Programming Center, 11400 Burnet Rd., Austin, Texas, 78758. Rogers is a staff programmer for FFST/2 and IBM SnapDump/2. He joined IBM in 1965 and initially worked on Manned Space Systems in Houston, Texas. He then served as the lead on RAS for Navy submarines in Massanas, Va., coded RAS on the Displaywriter, and led development of SNA management function for Communication Manager. He received an A.A. in computer science from San Jacinto College.

OBTAINING SNAPDUMP/2

IBM SnapDump/2 presently resides on OS2TOOLS for IBM internal personnel. Customers can get SnapDump/2 from the National Support Center (NSC) Bulletin Board System by phoning 404-835-6600. In Canada, phone 416-946-4244 (9600 bps) or 416-946-4255 (2400 bps).

SnapDump/2 is included with IBM Extended Services for OS/2, ESDTOOLS and Field Tool Warehouse.

SUPPORT FOR SNAPDUMP/2

User support for customers and SEs is provided through EQUAL/ASKQ. Keyword for entries should be "SNAPDUMP".

Defect support is handled through SNAPDUMP FORUM on IBMPC. Calls regarding defects will be answered within two weeks.



Customize your communications.

Quadron's development tools for OS/2 provide the flexibility you need for custom co-processor communications.

Match your protocols.

Use IBM ARTIC intelligent co-processor cards when you need extra ports or want to off-load protocol processing from your host computer. Then use Quadron software for custom communications with proprietary protocols, async, BISYNC, HDLC, X.25, and LAP-B. And if you need more than one protocol, no problem—just run them together on one card—at the same time.

IBM and OS/2 are registered trademarks of IBM Corporation.

Write & debug with ease.

You'll feel right at home with Quadron software. Just program in standard C, and use our messaging API to link OS/2 threads with tasks on the card. Then use our symbolic debugger and real-time analysis tools in OS/2 windows.

Protect your investment.

Perhaps best of all, our software is portable. Run what you write on any ARTIC card, on either ISA or Micro Channel. And if you later switch operating systems or platforms, just recompile.

Fit your applications.

Our easy-to-use software serves applications as diverse as telephone switching, financial transactions, satellite communications, and process control.

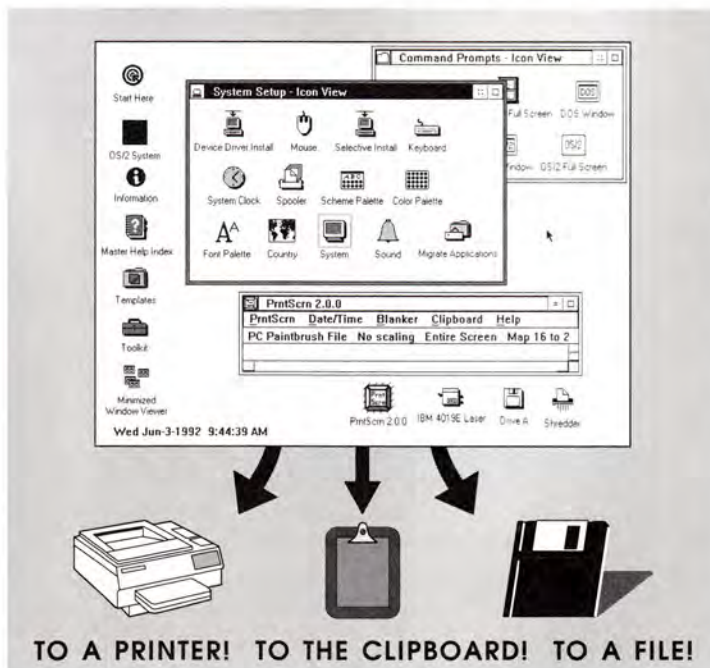
So call us. We've got an OS/2 communication solution—cards and software tools—ready to serve you today.



Quadron

Quadron Service Corporation
209 East Victoria Street
Santa Barbara, CA 93101
805-966-6424

CAPTURE YOUR SCREEN



PrntScrnm™ is THE One-Step Screen Image Utility for copying a graphics image from the Workplace Shell screen to a LAN or locally attached printer, to the Clipboard, or to a disk file. The screen images appearing in this publication were captured and processed using the **PrntScrnm** utility. The capture operation can be initiated from the keyboard (by pressing the print-screen key), from the mouse, or from another program. The captured image area can be the entire screen, current active window, current active client area, selected window, selected client area, or specified rectangular area. Disk file formats include PM Metafile, PM Bit Map, PCX, TIFF, GIF, and WPG. The "Color Mapping" feature provides the control needed to produce quality printed images from color screen images. **PrntScrnm** has clipboard Viewing, Printing, Exporting, & Importing capabilities for Bit Maps, Metafiles, and Text files. **PrntScrnm** includes a Screen Blanker and a digital Date & Time "information bar". **PrntScrnm** is priced at \$115 and is available from major software resellers, or contact MITNOR Software.

MITNOR Software
28411 E. 55th Street
Broken Arrow, OK 74014
Phone: (918) 357-1628
Fax: (918) 357-2869





32-Bit OS/2

Applications Printing Using OS/2 2.0



Michael Perks

by Michael Perks

This article describes how an application should print using OS/2 2.0. Most of the concepts and ideas presented here are also applicable to OS/2 1.2 and 1.3. Only 32-bit APIs have been used in the programming examples. The text and programming examples have been extracted from the OS/2 2.0 Technical Library: Programming Guide Volume III, Chapter 18. Corrections have also been made to the original text.

It is strongly recommended that you obtain the OS/2 2.0 Toolkit so the sample print program PRTSAMP can be examined for implementation detail of all the concepts presented here. The methods and recommendations should be followed whenever possible. This leads to consistency among OS/2 applications.

USER DIALOGUES

There are several stages involved in using the PM programming interface to print your application data. From a user-interface point of view, the following four dialogues must be provided. The exact order in which these dialogues are used will govern the order of program execution.

Page-Setup Dialogue

The page-setup dialogue lets users specify the form name or size required. Options on this dialogue can include margins on the page, header and footer strings, and whether duplex formatting is required.

The actual contents of the dialogue depend on the type of application. This page must specify formatting options that also are used to display to a screen.

Printer-Setup Dialogue

The printer-setup dialogue displays a list of queues available to the user. A job-properties button on this dialogue lets users query and modify the job properties for this job.

Font Dialogue

Most PM applications must allow users to specify the font (or fonts) required. Once they have chosen a printer, an option on the fonts dialogue lets users choose from device fonts, as well as system fonts.

Print Dialogue

The application should have a **Print** menu item on the **File** menu to invoke an application-specific print dialogue. It is recommended that the Shift-PrintScreen key be used as an accelerator for printing the client area. The PrintScreen key, unshifted, is used to capture and print a window or the whole screen.

PAGE-SETUP DIALOGUE

The page-setup dialogue is concerned with formatting options for the document. Users must be able to specify the form name, margins, and other application-specific formatting options such as page duplexing; that is, different formats for left and right pages in a multipage document. Figure 1 shows an example of a page-setup dialogue.

The application is responsible for storing the user-defined margins for the form and must not allow users to specify margins smaller than those returned by the printer driver.



FORMS SELECTION

If the user has not chosen a specific printer, the application can supply a list of standard form sizes; for example, letter, legal, ledger, A4, and A3. The application also must consider, at a minimum, a 0.4 inch (10 mm) margin on the form to be within the hardware clip limits of most printers. Table 1 shows the most common form sizes.

When a printer is chosen, it must be queried for the forms and hardware clip margins it supports using `DevQueryHardcopyCaps`. First however, a device context must be created. It is recommended that a `OD_INFO` context be used because `OD_QUEUED` can result in the creation of a print job. Figure 2 shows a sample code fragment.

`DevQueryHardcopyCaps` returns one `HCINFO` structure for each form. The contents of the structure are:

- ASCII name of the form (for example, Letter)
- Width and height (in millimeters) of the paper
- Clipping limits (in millimeters) of the paper
- Number of pels between clipping limits
- Whether this form is the one installed currently on the device, or whether it is selectable from another paper bin.

Then, the list of forms can be displayed to the user. The form preselected in the list should be one of the forms marked with the `HCINFO` structure flag `HCAPS_CURRENT`.

An application should indicate to the user which forms are currently installed in the printer by including the `HCINFO` structure flag `HCAPS_SELECTABLE`. Then, users can decide whether they want to print quickly on a form available from the printer or to install a different paper tray in the printer.

PRINTER-SETUP DIALOGUE

In a printer-setup dialogue, an application should offer a list of available printer objects and let the user select one. (The list of printer

objects actually is a list of queues.) If none is available, an appropriate message must be displayed. An application must query the list of available printer objects each time the printer-setup dialogue is displayed because the user might have created or modified the printer configuration while the application was executing. Figure 3 shows an example of a printer-setup dialogue.

If the document was formatted for a particular device and the user selects a different printer, the application must ask the user's permission before reformatting the document for the new printer.

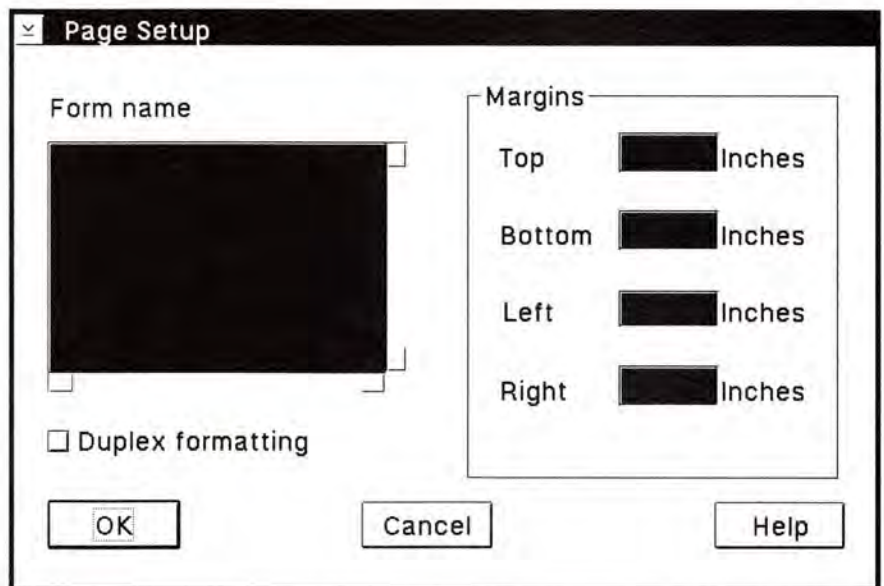


Figure 1: Application page-setup dialogue

COMMON FORM SIZES

Form Name	Size in inches	Size in mm
Letter	8.5 x 11	216 x 279
Legal	8.5 x 14	216 x 356
Ledger	11 x 17	279 x 432
A4	8.3 x 11.7	210 x 297
A3	11.7 x 16.5	297 x 420

Table 1: Common form sizes

Use `Sp1EnumQueue` to query the list of printer objects; printer objects essentially are spool queues. `Sp1EnumQueue` returns a list of queues and information about each queue on the local workstation in an array of `PRQINFO3` structures. It also returns information about local workstation queues that reference network print queues.



```

PPRDINFO3    pprd3Device;           /* From SplQueryDevice */
PPRQINFO3    pprq3Queue;           /* From SplQueryQueue */
PSZ          pszTmp;               /* Temporary pointer */
HDC          hdc;                  /* Device context handle */
DEVOPENSTRUC dopData;              /* DEVOPEN structure */
LONG         clForms;              /* Number of forms */
                                   /* The device */
PHCINFO      pchinfo;              /* Forms information */
                                   /* structure */
ULONG        ulrc=DEV_OK;          /* Return code */

/* Fill in data for devopendata for OD_INFO */
dopData.pszLogAddress = pprd3Device->pszLogAddr;
pszTmp = strchr(pprq3Queue->pszDriverName, '.');
if (pszTmp)
    *pszTmp = '\0';
dopData.pszDriverName = pprq3Queue->pszDriverName;
dopData.pdriv = pprq3Queue->pDriverData;

/* Open the information device context */

hdc = DevOpenDC ( (HAB)0,
                 OD_INFO,           /* Type */
                 "*",               /* Default token */
                 3L,                /* Count */
                 &dopData,          /* Pointer to data */
                 (HDC)0);           /* Comp dc */

/* Query number of forms available on the device */
clForms = DevQueryHardcopyCaps(hdc,
                                OL,   /* Start at beginning of list */
                                OL,   /* Get number of forms */
                                NULL);

/* Allocate memory block for forms */
if (DosAllocMem((PPVOID(&pchinfo), clForms*sizeof(HCINFO), fALLOC))
{
    DevCloseDC(hdc);
    return(DEV_ERROR);
}

/* Query forms data */
ulrc = DevQueryHardcopyCaps(hdc,
                             OL,   /* Start with first form */
                             clForms, /* Query all forms */
                             pchinfo); /* Structure to hold returned */
                                     /* data */

/* Close the information device context */
DevCloseDC (hdc);                 /* Close the information */
                                     /* device context */

/* Now use forms information in pchinfo */

```

Figure 2: Querying the printer using DevQueryHardcopyCaps



Because the number of queues might vary for each application use, it is essential to allocate sufficient storage to hold the data returned by `Sp1EnumQueue`. Usually the application issues the query twice. The first time, the application determines the necessary size of the information buffer. After allocating a memory block, the second query actually retrieves the information.

The queue description (returned in the structure `PRQINFO3` field name `pszComment`) is the printer object title. This command is much more familiar to users than the queue name, which is displayed only on the view settings page of a printer object. Therefore, the printer-setup dialogue should show the queue descriptions instead of the queue names.

`Sp1EnumQueue` returns information about the queue that might influence the user or application's decision to print to this queue; for example, the number of jobs already in the queue. `Sp1EnumQueue` also returns the default job properties for the queue. This data can be used by the application for the job-properties button on the printer-setup dialogue.

JOB-PROPERTY CONSIDERATIONS

Job properties are options on a per-job basis; for example, orientation, resolution, and form selection. The job-properties dialogue is displayed by the printer driver. Each driver has a different dialogue, depending on the capabilities of the printer. The job properties are held in a printer-driver-specific format in the `abGeneralData` field of the `DRIVDATA` structure.

Job properties from one printer driver should not be given to another printer driver. The job properties probably will not be understood, and the printer driver will either return an error or substitute some default job properties. Then, the user will see a change in job properties stored previously.

Users should be given the opportunity to change the job properties before the job is printed. This can be achieved by supplying a job-properties push button on the printer-setup dialogue.

Retrieving Job Properties

The application can retrieve job properties from the following locations:

- Previously saved job properties with the document
- Current application session job properties
- Default job properties from the queue (from `pDriverData` in the `PRQINFO3` data structure)
- Printer driver device defaults (from `DevPostDeviceModes` using the `DPDM_QUERYJOBPROP` flag).



Figure 3: Application printer-setup dialogue

If no job properties were saved with the document, there may be job properties being used by another document that is to be printed to the same queue.

If job properties still cannot be found, the default job properties stored with the queue can be used. The user may not set up any default job properties for the queue. Finally, query the printer driver for its device defaults using `DevPostDeviceModes` with the `DPDM_QUERYJOBPROP` flag.

The sample code in Figure 4 shows how to display the job-properties dialogue. All the parameters required are available from the `PRQINFO3` structure returned by `Sp1EnumQueue` or `Sp1QueryQueue`.



```

200 #define INCL_DEV
    #define INCL_DOS
    #include <os2.h>
    #include <memory.h>

    {
        ULONG          ulrc=FALSE;
        HAB             hab;
        PPRQINFO3       pprq3Queue; /* From SplEnumQueue or SplQueryQueue */
        PSZ              pszDriverName,pszDeviceName,pszTmp;
        HDC             hdc=NULL;
        LONG            cbBuf;

        /* Use the first device name in the PRQINFO3 structure */
        pszTmp = strchr(pprq3Queue->pszPrinters, ',');
        if (pszTmp)
            *pszTmp = '\0';

        /* Use just the driver name from the driver.device string */
        pszDeviceName = strchr(pprq3Queue->pszDriverName, '.');
        if (pszDeviceName)
        {
            *pszDeviceName = '\0';
            pszDeviceName++;
        }

        /* Check size of buffer required for job properties */
        cbBuf = DevPostDeviceModes( hab,
                                    (PDRIVDATA)NULL,
                                    pprq3Queue->pszDriverName,
                                    pszDeviceName,
                                    pprq3Queue->pszPrinters,
                                    DPDM_POSTJOBPROP
                                    );

        /* Return error to caller */
        if (cbBuf<=0)
            return(cbBuf);

        /* Return BUFFER TOO SMALL error to caller */
        if (cbBuf > pprq3Queue->pDriverData->cb)
            return(DPDM_ERROR);

        /* Display job properties dialogue & get updated job props from driver */
        ulrc = DevPostDeviceModes( hab,
                                    pprq3Queue->pDriverData,
                                    pprq3Queue->pszDriverName,
                                    pszDeviceName,
                                    pprq3Queue->pszPrinters,
                                    DPDM_POSTJOBPROP
                                    );

        return(ulrc);
    }

```

Figure 4: Displaying the job properties dialogue

FONT DIALOGUE AND DEVICE FONT CONSIDERATIONS

There are two types of fonts:

- System fonts, which can be used for displays and printers, and therefore, are generally more flexible.
- Device fonts, which are specific to a particular device. Device fonts for printers can be built-in or installed with font cartridges by a user. Device fonts can also be available on disk (also termed soft fonts) and downloaded to the printer as required by the printer driver.

The advantage of device fonts is that they are usually printed in the highest resolution of the device and are faster than system fonts.

Some printer drivers, in particular the HP LaserJet (LASERJET) and the LaserPrinter (IBM4019) printer drivers, provide an intermediate solution. An option on the job-properties dialogue enables system fonts to be downloaded to the printer as soft fonts.

An application must provide users with the ability to choose fonts and, in particular, to choose a device font over a system font to achieve better performance. A standard font dialogue box should be used, such as `WinFontDlg`.

When an application uses device fonts and the output mixes text and graphics, the device fonts are clipped per character. System fonts give more precise clipping to the pel. Figure 5 illustrates the results from clipping two lines of text: one generated in a system font, the other in a device font.

The available device fonts can be queried, using `GpiQueryFonts` in either an `OD_INFO` or `OD_QUEUED` device context. Device fonts are returned with negative `lMatch` numbers. Then the appropriate logical font can be used after issuing `GpiCreateLogFont`.

If, during the printing process, a logical font is not created, the printer driver uses its default font for the printer, which is usually 12-point Courier.

There are two design choices for device fonts. First, use system fonts for the display. When

printing, query the printer driver and attempt to choose a device font that closely matches the system font. If no match is found, then print using the system font. A device font's metrics and character spacing might not match exactly, so the printed output might not be exactly the same as the display.

Second, use a printer font if the user selects one. Determine the metrics and find a system font that shares the same characteristics (for example, a style such as Courier or Serifed). The character string to be displayed is sent to the printer driver and the intercharacter spacing is returned, using `GpiQueryCharStringPos`. The individual character positions are used when the string is sent to the display, using `GpiCharStringPos` with the `fOptions` parameter of `CHS_VECTOR`. While the display output can take longer to display to the screen, true WYSIWYG (What You See Is What You Get) is achievable.

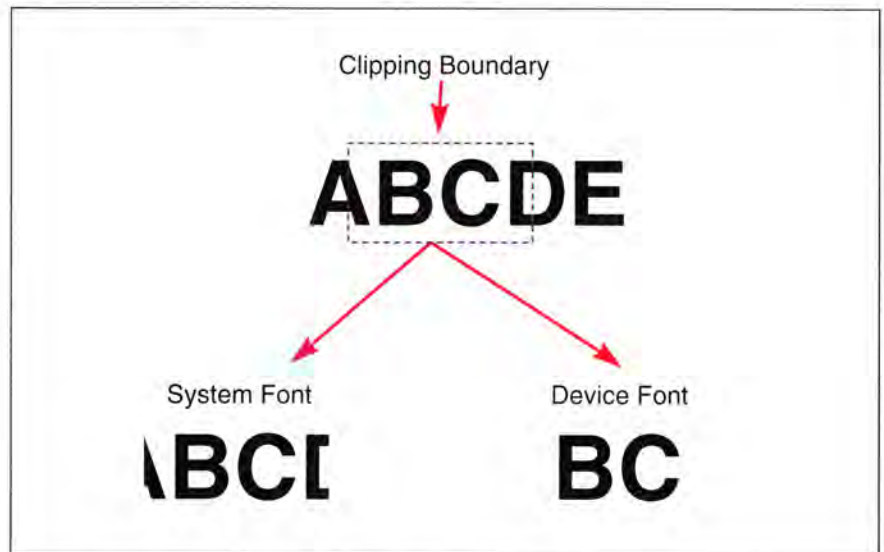


Figure 5: Character clipping results for device and system fonts

PRINT DIALOGUE AND PRINT PROCESSING

The print menu must display a dialogue to allow the user to specify the number of copies, start page, end page, or any other application-specific print options. A confirm button initiates the print. Figure 6 is an example of a print dialogue.

The **Printer** entry field is the read-only name of the printer object (queue) the user chose from the printer-setup dialogue. The **Preview**



checkbox is used to preview the printed output on the screen. This can be achieved by opening an `OD_MEMORY` device context to the printer driver and drawing the picture into the bit map. A `Bit Blt` to the screen shows the resulting page. System fonts should be used instead of device fonts.

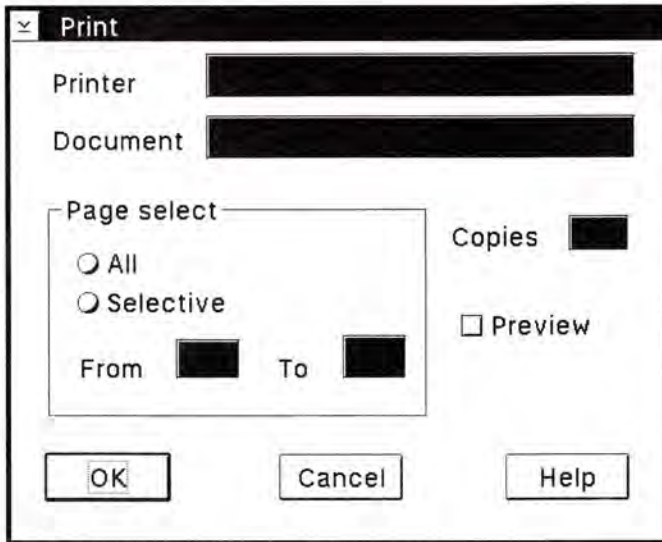


Figure 6: Application-Print dialogue

Once the user has initiated the print, a PM application must execute the following steps:

1. Create a new thread.

Then, on this new thread:

2. Open a device context.
3. Associate a presentation space.
4. Start the print job.
5. Query the device capabilities.
6. Format the page.
7. Start the next page.

Repeat Steps 6 through 7 for each page.

8. End the print job.

For another print job, repeat from Step 4.

9. Disassociate the presentation space.
10. Close the device context.

The following sections describe each of these steps.

Creating a New Thread

Once the print process has been confirmed by the user, the application must use a separate thread to perform the actual printing. This maintains user responsiveness and avoids displaying the hourglass pointer. While the printing is in progress, the application should dim certain menu options, such as drawing. Other options, such as page down, zoom, or help must still be available.

Opening a Device Context

Before you can send data to a printer using device functions, you must open a device context with `DevOpenDC`. The parameters of `DevOpenDC` are influenced by the queue and job properties the user has chosen previously.

`DevOpenDC` accepts the following parameters as input:

- **hAb**—the anchor-block handle from a `WinInitialize` call.
- **lType**—the type of device context. This is always `OD_QUEUED` to specify a queued device context.
- **pszToken**—the device-information token. Always use an asterisk (*) for this parameter to force the system to get device information from the `pdopData` parameter.
- **lCount**—the number of data elements supplied in the `pdopData` parameter. The minimum number of parameters for a queued device context is four.
- **pdopData**—the device-context data area. This is a pointer to a structure of the type `DEVOPENSTRUC`. The `DEVOPENSTRUC` structure contains all the data needed to define a device context. At least the first four structure members must be provided for queued device contexts.
- **hdcComp**—the compatible device-context handle. This must be `NULL` for a device context of the type `OD_QUEUED`.

`DevOpenDC` returns a device-context handle of the type `HDC`. The handle is used in other



functions beginning with the Dev prefix, and for GpiCreatePS and GpiAssociate.

The DEVOPENSTRUC structure contains all the data needed to define a device context. Its individual elements are described below. At least the first four structure members must be provided.

- **pszLogAddress** is the name of the queue. It is the pszName field of the PRQINF03 structure. For device contexts of the type OD_INFO, pszLogAddress is the port name. This can be retrieved by calling Sp1QueryDevice using the pszPrinters device name. The pszLogAddr field in the PRDINF03 structure is the port name.
- **pszDriverName** is the character string identifying the printer driver, for example, LASERJET. The pszDriverName field of the PRQINF03 structure associated with the required print queue gives the driver and device name, separated by a period, for example LASERJET.HP LaserJet IIID. The pszDriverName field can contain only the name up to the period, for example LASERJET.
- **pdriv** is a pointer to the job-property data returned by the printer driver from DevPostDeviceModes or the default job properties from pDriverData in the PRQINF03 structure. The DRIVDATA structure contains the particular device name to be used. Therefore, it is a programming error to set this parameter to NULL.
- **pszDataType** is a data type. It is recommended that "PM_Q_STD" be used here.
- **pszComment** is an optional character string the printer object displays to the user in a job-settings notebook. The application should include its own name in this comment string. The job title text is derived from the document name.
- **pszQueueProcName** is an optional queue processor name. The queue processor (also termed "queue driver") name is available from the pszPrProc field of the PRQINF03 structure. The default queue processor provided by the operating system is PMPRINT. The user also can install a queue processor (PMPLLOT) used to provide reverse clipping for vector devices such as plotters.
- **pszQueueProcParams** are optional queue processor parameters. They can include information such as the number of copies you want to print and the size of the output area on the printed page.
- **pszSpoolerParams** are optional spooler parameters separated by spaces. They are used for scheduling print jobs, such as the form names that identify the paper to be used, for example, FORM=LETTER.
- **pszNetworkParams** is an optional parameter that can be used to specify network options, for example, USER=JOESMITH.

Associating a Presentation Space

To print, a device context must be associated with a GPI presentation space. In most circumstances, a presentation space already exists for the screen. In such cases, the presentation space can be disassociated from the window device context and associated with the printer device context by using GpiAssociate.

If a presentation space does not exist, or a different one is to be used, the application must issue GpiCreatePS and use the GPI ASSOC flag to associate the printer device context.

Device-Independence Considerations

The operating system supports true WYSIWYG. The same GPI functions the application used to create the picture or document on the display screen can be used to create the output on a printer. This is the major advantage of device independence.

An application must be designed around the options provided by the PM programming interface to ensure device independence, as a display (VGA) typically is 96 dpi, while a printer typically is 300 dpi.

Some application design considerations apply. Use DevQueryHardcopyCaps to determine the size of the form and the maximum printable area. Create a presentation space with a presentation page size of 0 to ensure that the maximum window and page size are used. Use device-independent coordinates, such as LO ENGLISH or TWIPS. If you program in pels, transferring your output from screen to hardcopy will compress the picture.

An application must be designed around the options provided by the PM programming interface to ensure device independence.



Starting a Print Job

The application signals the beginning of a print job by using `evEscape` with the escape parameter `DEVESC_STARTDOC`. The application must specify a document name, usually related to the name of the file being printed. Any GPI calls made before this `DevEscape(DEVESC_STARTDOC)` are ignored.

Querying the Device Capabilities

`DevQueryCaps` can be used to provide over 40 pieces of information about a specific device; for example, default character box size, horizontal and vertical resolution, or number of device-specific fonts. Knowing the device has only 1-bit color support (monochrome) might influence the application to use line style and patterns instead of colors for output.

Formatting the Page

Before print processing actually starts, it is recommended that an application modal dialog be used to show page progress and allow the user to cancel the print job. If the user presses cancel on this dialogue, the application must issue `DevEscape` with the parameter `DEVESC_ABORTDOC`. This command enables the spooler and printer driver to clean up. A job is not created in the queue.

Usually, the output on a page is generated by a series of GPI calls from the application. The application must be able to decide when to move to the next page in a multipage document by calculating the size of character strings or graphics written to the presentation space.

Starting the Next Page

When the application has finished processing a page, it must issue `DevEscape` with the `DEVESC_NEWFRAME` parameter. The progress dialogue must be updated.

The `DevEscape DEVESC_NEWFRAME` can be used to generate blank pages. The printer driver always issues a page eject whenever this `DevEscape` is received. Issuing the `DEVESC_NEWFRAME` escape does not reset any attributes or fonts. However, the bounds and any clipping regions are reset.

Ending the Print Job

The end of a print document is signalled by the application's issuing `DevEscape` with the `DEVESC_ENDDOC` parameter. The spooler returns a job identifier that can be used for job manipulation.

Disassociating the Presentation Space

When the print job is complete, the presentation space can be disassociated from the printer device context using `GpiAssociate` with a `NULL` parameter. Then the presentation space can be reassociated with the display device context.

Closing the Device Context

A device context is closed using `DevCloseDC`. The print thread must signal print termination to the main processing loop. The main loop can terminate the print thread and dismiss the `Page Progress` message box.

An application can display a message box to the user indicating a successful print and show the job ID returned by the spooler.

PRINT TO FILE CONSIDERATIONS

Print to file means that the print data destined for a printer is stored in a file instead. The advantage is that the file can be used later, possibly on a different machine or environment, and the data can be printed without requiring the original application.

The preferred method to print to file is for the user to specify **Print to file** on the **Output** settings page of a printer object. When a print is requested, the system displays a dialogue in which the user can enter a file name. The application does not have to be aware of the print to file; the user has full control of the system.

NETWORK PRINTING CONSIDERATIONS

The PM architecture requires a printer driver during the print job creation process to deal with queries such as `DevQueryHardcopyCaps` and to provide the job-properties dialogue by way



of `DevPostDeviceModes`. To do any printing across a network, a locally installed printer driver is required. In some cases, such as when the network server does not run PM, this is an advantage because all the conversion-to-printer specific commands can be done on the requestor.

The printer object in the workplace provides seamless access to network printers; the user need only install the appropriate printer driver.

The application programmer is also helped. A network printer accessed by the user has a hidden shadow on the requestor. The job properties for the shadow printer object can be altered and are stored on the requestor. This capability enables the user to configure one or more variations on the original configuration without requiring intervention from a system administrator to create many network printer objects, each with a slightly different default job-property configuration.

The shadow printer object is created on the requestor by creating a local queue and local device with no port connection. The application can enumerate these queues using `SpEnumQueue`. The only difference is in the `PRQINFO6` structure; there are two extra fields not in the `PRQINFO3` structure. These two extra fields, called `pszRemoteComputerName` and `pszRemoteQueueName`, contain the name of the remote server and queue. The spooler uses these fields to automatically reroute print data submitted by an application to the local shadow to the appropriate network queue.

The printer object creates a local shadow only when the network print object has been used or modified because there could be a large number of network printer objects, and the spooler does not know which one the user wants to use. The local shadow of a network printer object is created in these cases:

- When a file is dragged and dropped on the network printer object
- When the network printer object is moved, copied, or shadowed outside its original server folder
- When the printer or job properties are modified.

DRAG/DROP PROTOCOL CONSIDERATIONS

When an application data file is dropped on a printer object, the application can receive a `DM_PRINTOBJECT` message if the application is running currently.

One `DM_PRINTOBJECT` parameter gives a `DRAININFO` structure that describes the object dropped. The other parameter gives a `PRINTDEST` structure that contains all the parameters required to issue `DevPostDeviceModes` and `DevOpenDC`.

The `PD_JOB_PROPERTY` flag indicates to the application that the user has requested a job-properties dialogue before printing. If this flag is not set, the application must not display the job-properties dialogue; it must use the job properties passed in the `PRINTDEST` structure. The rest of the printing process follows the procedure described previously.

Michael Perks, IBM Corp. Internal Zip 1510, 1000 N.W. 51st St., Boca Raton, Fla. 33431. Perks is a project programmer in OS/2 Printing Development. He is the designer for the OS/2 2.0 Print Subsystem and the OS/2 2.0 LAN independent shell. He joined IBM in 1984 and has been working on OS/2 since 1986. He received a B.Sc. from Loughborough University of Technology in Loughborough, U. K.

REFERENCES

OS/2 2.0 Technical Library: *Programming Guide, Volume III*. (IBM Doc. S10G-6495)

Bernath, David A. "File and Font Dialogs: Standardized Selection Techniques," *IBM Personal Systems Developer* (Winter 1992): 18-26.

Reich, David E. "Programming Printing Using OS/2," *IBM Personal Systems Developer* (Winter 1992): 85-95.



32-Bit OS/2

Where, Oh Where, Did the Print Manager Go?

Printing with the OS/2 2.0 Workplace Shell

by Abby Gelles Knott

If your software ran under OS/2 1.3, your customers might have needed help with printer setup or job management. OS/2 2.0 provides a more intuitive print-subsystem interface than the procedure-oriented, menued OS/2 1.3 Print Manager. The OS/2 2.0 Workplace Shell empowers your application customer to make some of the choices you used to program. By opening the various views onto simple desktop objects, anyone can tailor a configuration, check the progress of print jobs, and solve problems. In fact, the OS/2 2.0 on-line help information urges your customer to do so.

This article will familiarize you with your customer's view of printing under OS/2 2.0. In the previous article, "Application Printing Using OS/2 2.0," Michael Perks provides many of the programming details you need to coordinate with the shell in protecting your customer's preferences. Perks designed the shell you are about to explore.



Figure 1: Print object icon on the desktop

OVERVIEW—WHAT'S NEW?

Did your customer have difficulty understanding the OS/2 1.3 Print Manager queues? Queues have little importance to the user of the OS/2 2.0 shell. By contrast, you, the developer, focus more on queues (and less on devices or ports) than you might have previously. Queues have a new face and are tightly coupled with their fellow print mechanisms, printer drivers, and ports. A set of these items is packaged under an icon that looks like a printer, called a printer object. Figure 1 shows the OS/2 desktop with one printer object icon in its closed state.

Each variation in the setup can have its own printer object. You might think of each printer object on a desktop as a mini-Print Manager. Figure 18-1 in the *OS/2 Programming Guide* shows how flexibly the printer-object mechanism permits queues and devices to support one another. Thanks to OS/2 2.0 LAN independence, your customer simply manipulates another printer object to print remotely, while you use the same APIs to communicate with a network printer object as you would with a local printer object.

WHY SHOULD YOU BE CONCERNED WITH THE SHELL?

Your program views the printer object as a print destination and does not concern itself with a physical device. The printer object looks like a printer even if it represents a setup that prints to a disk file. This abstract object is the printer your user selects when you ask, "Where do you want to route this job?" Your application should identify printer objects by their descriptive names, which might be

"Harry's printer," "My printer for PostScript Jobs," not by their internal names, such as LPT1Q1. (You list the name pointed to with `pszComment`, rather than `pszName` for the queue, from PRQINF03.)

To the user, printer objects combine to form a personal setup. Because an application that ignores that setup or overrides it without first notifying the user can cause consternation, the remainder of this article provides a quick overview of that setup. To help you understand the relationship between objects in the shell and the printing thread in your application, this article references Perks' article and "Print Job Submission and Manipulation" (IBM, *OS/2 Programming Guide*, 1992).

SETUP

Initial Installation

Selecting a printer during operating-system installation prompts the system to install the appropriate printer driver and create a printer object with that printer driver as its default. Installation also provides objects, blatantly displayed, that open onto on-line information. Within that information are step-by-step instructions for installing additional printers and drivers.

Customization Settings and Properties

To adjust printing setup, your user changes printer-object settings. When the printer object is open to its settings view, the shell displays a spiral-bound notebook with a separate tabbed page for each category of setting. This section explains which of the user's choices on these pages influences your application.

Output

The output settings page, shown in Figure 2, enables a user to select and configure physical ports or to initiate a print-to-file command.

Port Installation

A default system has parallel connections LPT1 and LPT2 and serial connections COM1 through COM4, but the pop-up menu for any port object on this page includes an option for installing other flavors. No matter what ports your user has, your program does not capture a logical

port address (LPT1). As you will see, a printer object can have multiple logical addresses or no port assigned to it.

Port Configuration

To adjust the timeout period for a device, a user opens (double-clicks on) the connecting port object inside the **Output port** field. A window opens, exposing a **Timeout** setting field, which replaces "Printer Timeout Transmission Retry" in the setup-printer dialogue of the OS/2 1.3 Print Manager. This port setting takes on increasing significance with the proliferation of PostScript printers. Opening the object for a serial (COM) port also exposes



COMPONENTS OF AN OS/2 PRINT SUBSYSTEM

A system needs the following components before printing can occur:

- The **printer object** is an icon that represents one print destination. The shell enables a user to drag any printable object to this object for instant queuing. The operating system enables your program to view each printer object as an integrated print destination.
- The **queue driver** software assembles and distributes separate print jobs.
- The **printer driver** software and information controls print-job setup and ultimately converts jobs to the format a particular printer model can interpret. Another name for the printer driver, as it applies to your Presentation Manager printing API, is presentation driver. Some of the developer documentation uses the latter term.
- The **output device** is usually a printer that takes an image out of computer memory and imprints it on a tangible medium, such as paper. Actually, your program might interface with a printer, a plotter, a device that produces nonpaper media (such as film), a communications or fax modem through which you send print-formatted information, or a disk file containing a printer-ready version of a document. Because the print subsystem can print to a file, device hardware is optional.
- The **port** is a 25- or 9-pin connector used to attach printers, plotters, and communications devices.
- The **spooler** is where print jobs are actually held together in one area of the disk, the spooler path, which houses the print files for all printer objects. The shell contains a separate Spooler object, which represents the controlling program that manages queues. This object provides a means to change the location of the system spool file or to disable spooling for direct printing.



adjustable communications parameters such as baud, word size, handshake, and so on.

Pooled Printers

If a system has multiple printers, each connected to a different port, the system owner can improve throughput by selecting all

Output to File

A person who occasionally prints to a file creates a separate printer object and puts a check mark in the **Output to file** field. Any job routed to this particular printer object invokes a print-to-file dialogue that prompts for a file name.

Printer Driver

This notebook page, shown in Figure 3, contains objects from which a user accesses the dialogues for adding printer drivers and for configuring printer properties, fonts, and default job properties. It is also on this page that a user designates a specific driver for use in printing.

Default Driver

If the system has multiple printer drivers installed, the user can change the driver assigned to this printer object at any time. Different drivers can potentially give the same output different looks, even when they are printed by the same physical device. Preferably, the user will create a separate printer object for each driver desired. The on-line information points out these subtleties in configuration and explains how to achieve them.

Printer Properties

A printer driver has a device-specific printer-properties dialogue for indicating the actual physical setup of a device, such as number of trays, forms used, and cartridges loaded. The on-line information instructs your user to make printer-driver selections after installing a driver and maintains these fields appropriately to match changes in loaded forms, font cartridges, plotter pens, or other resources. When your software allows the user to choose a document form and fonts, it references the printer-properties information to verify the availability of these job-specific requests.

Opening the object in the **Default driver** field accesses the printer-properties dialogue. (Printer properties are really printer-driver settings, but the dialogues retain the look and names they had under OS/2 1.3.) Figure 4 shows a printer-properties dialogue with fields

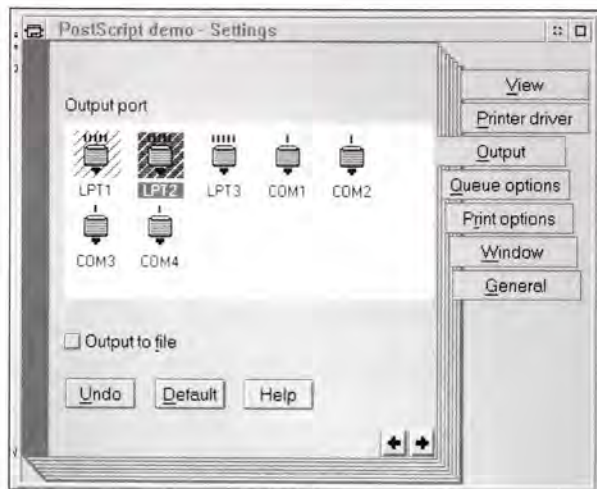


Figure 2: Output settings notebook page

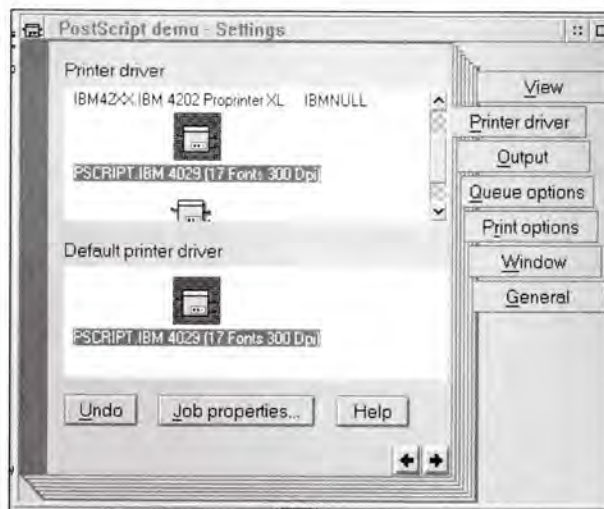


Figure 3: Printer driver notebook page

those ports in the **Output port** field. All jobs queued at the printer object have several output possibilities and each, in turn, is sent to the first available port. Consider the benefits of this setup, called pooled printers, to the administrator of a LAN printer server. Again, since the user can pool printers, you do not want your application to query a logical port address or route output to it.

that pertain to the specific capabilities of a particular printer model.

Device (Printer) Font Installation

Printer properties include a dialogue for installing device fonts, including extended font character sets and patterns. Today's lay and commercial public has some savvy concerning the influence of fonts in calling attention to documents. Consult Perks' article or the *OS/2 Programming Guide* section "Font Dialog and Device Font Considerations" to find how your program uses logical fonts. If you use logical fonts from GPI APIs within your program, your user can change font selection from the shell without having to also adjust a document.

Job-property Defaults

Each printer object has defaults for such per-job characteristics as orientation, resolution, form selection, and one- or two-sided printing. Default job properties provide the per-job setup your application queries (with `DevPostDeviceModes`) and displays, then invite the user to change them. The **Printer driver** settings page has a **Job properties** button control for accessing these default settings.

Queue Options

This settings page, shown in Figure 5, provides a way to change queue drivers or to specify `PM_Q_RAW` data type.

Queue Processor (or Driver)

Most printers and plotters use the standard `PMPRINT` software queue driver, but the pop-up menu for `PMPRINT` has an option for installing a different one. Selecting the added driver in the **Queue driver** field activates it.

PM_Q_RAW (Printer-specific Format)

If a printer object has a check mark in the **Printer-specific format** field, all its queued spool files contain `PM_Q_RAW` data or printer-specific control characters. By contrast, the more concise metafile format (or `PM_Q_STD` data type) holds a description of job setup that needs further interpretation by the printer driver. The selection for this field is indicated internally as data type (`PRQINFO3 sfType`), as explained in detail in Perks' article and the

OS/2 Programming Guide. The system automatically converts a `PM_Q_STD` file to `PM_Q_RAW`, when required by the printer-object setup, freeing your application from the necessity to indicate a data type.

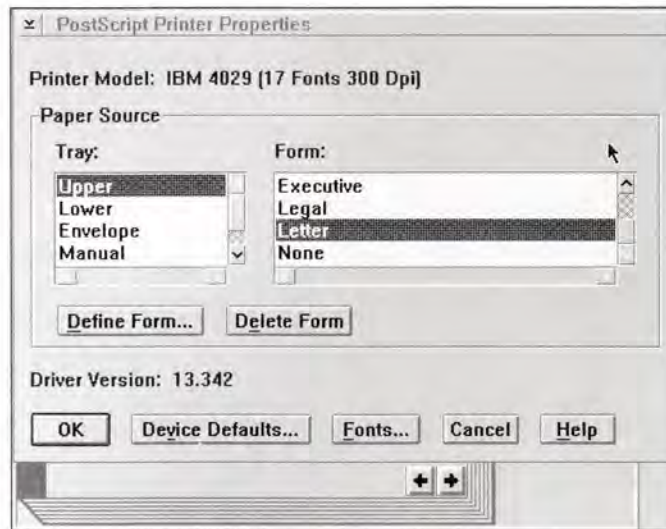


Figure 4: Printer properties dialogue

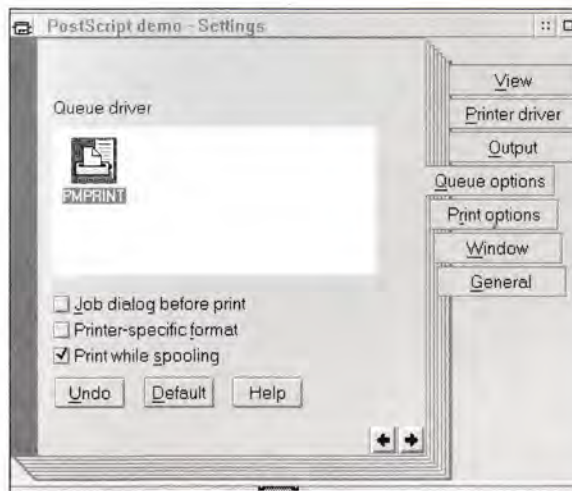


Figure 5: Settings notebook page

Job Dialogue Before Print

A user can also request to have the job-properties dialogue displayed for modification each time a print request involves this particular printer object. The dialogue does not appear in conjunction with a job printed from your application and does not affect jobs created by your application. It is provided for people who print directly on the desktop, using the drag-and-drop method.



THE DEFAULT SYSTEM PRINTER

In the past, a Print Manager™ menu option called **Application default** was used to change the queue, serving as the system default. Every OS/2 2.0 printer object's menu has an option called **Set default** that performs the same function. Despite its former name, this default has little influence on your application. It specifies the device for output generated by pressing the Print Screen key or by selecting a button to print a Help topic.

print without queuing, your user can disable the spooler for the entire system. If you use an **OD_DIRECT** device context, you (and your customer) forfeit all these nice setup opportunities provided with the printer object. You also take away your user's opportunity to manage a job from the open printer-object window. And, of course, you foil the efficiencies of spooling by tying up the printer with your direct-output job.

JOB MANAGEMENT

You guessed it, jobs are objects. The user can directly manipulate them or select menu options for them. Once the spooler assigns the job to its printer object, opening the printer object to its default icon view displays a window where the pending job icons live. In reality, these jobs rarely stay on the queue long enough for anyone to inquire about or alter them. If you want to check the state and setup for job objects while testing your application, use the pop-menu of the printer object, the **Change status** option, to hold the printer object.

With the job-object menu and settings, any shell user can alter printing order or job priority; copy, delete, or hold and release jobs; and even override area and fit. Double-click on a job to view its content; press the second mouse button to display a pop-up menu from which you can open to the settings view. As application developer, you will find the settings pages of a job indicate how well your application accomplished its job-submission setup. Each tabbed page of the job settings has a **Help** button, in case you do not find the fields self-explanatory.

Perks' article and the *OS/2 Programming Guide* tell you how to display job-properties defaults to give your user an opportunity to override them for a single job. To ascertain whether your software correctly spooled the overridden properties, double-click on the printer-driver icon shown in the upper right of the job's **Submission-data** settings, shown in Figure 6. In this context, the printer-driver settings pertain to job properties, not printer properties.

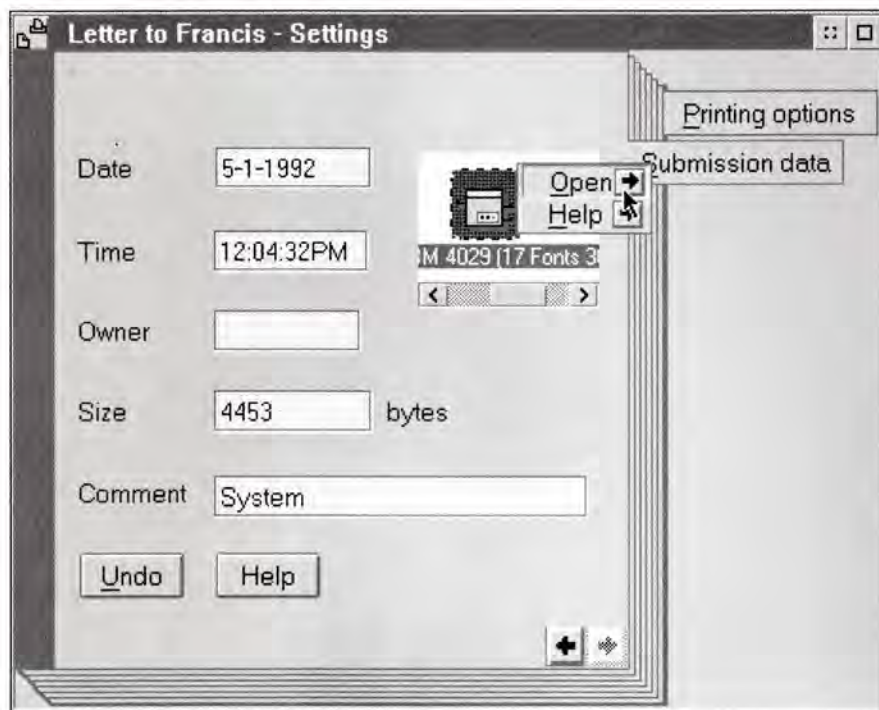


Figure 6: Submission-data settings

Spooler Options

Path opens the **Spooler** object to see this sole setting, provided as a performance enhancement. If you write an application that relies on large **PM_Q_RAW** queue files or one with which you expect your customer to create a large volume of simultaneously queued jobs, suggest in your documentation that the customer change the path in this field, perhaps to a separate disk.

Direct printing indicates that there is no means to set up a printer object for direct printing. To



Abby Gelles Knott, Information Developer, Computer Task Group—Information Media Practice (CTG-IMP), 3302 Poolside Dr., Greenacres, Fla. 33463, 407-967-8158. Knott has a master's degree in computer methodology and has been programming, writing about, and training people to use computers for over 20 years. She produced on-line information for the OS/2 2.0 print subsystem, network interface, templates, and color palette, and was a developer of on-line Windows- and DOS-compatibility information. She also designed the OS/2 2.0 Information and Planning Guide.

RESOURCES

About the Interface

Two on-line procedural guides have a separate icon directly on the desktop:

1. Start Here topics installing printers and printing.
2. The Master Help Index topics printing, queue, port, and spooler.

Consulting Start Here and the Master Help Index gives you, the developer, a feel for your users' expectations concerning printing.

About Programming to the Interface

These references cover the graphics interface and printing. Use them not only for issuing your print job, but also for determining which APIs you need to ensure the appropriate font and graphic transformations:

"Chapter 18: Print Job Submission and Manipulation," *The OS/2 Programming Guide Volume III: Graphic Programming Interface*.

Presentation Manager Programming Reference Volume III (available on line).

Developer's Toolkit for OS/2 2.0, sample program PRINT.

Perks, Michael. "Application Printing using OS/2 2.0," *IBM OS/2 Developer* (Summer 1992): 42-51.

Reich, David E. "Programming Printing Under OS/2," *Personal Systems Developer* (Winter 1992): 85-95.

TIPS AND MISCELLANY

- **Port timeout.** If you anticipate PostScript printing, advise your customer to set a minimum of 120 seconds as the timeout period to prevent frequent off-line messages.
- **Reverse clipping.** If your drafting software has no option to hide the background lines of solids, advise your customer to install the PMPLOT.DRV printer driver and PMPLOT.QPR queue driver, both of which are supplied with OS/2 2.0. The user can retain a separate printer object with these two selected. This printer object must have the plotter driver for the model and PMPLOT.DRV selected in the Default printer driver settings field. For more details about reverse clipping, select the topic "plotter, reverse clipping considerations" from the on-line Master Help Index.
- **Network printing.** Device-independent (PM_Q_STD) metafiles are smaller than printer-specific format (PM_Q_RAW) files by a factor of five. Obviously, the smaller metafile will travel over the network more efficiently. If a network server supports OS/2 printer drivers, the system automatically gives your customer optimum performance by using the standard data type. If the network does not support PM metafiles, the local OS/2 seamlessly converts your print-job stream to specific (RAW) format required by the network printer. For the network printing data structures, see the *OS/2 Programming Guide* section "Network Printing Considerations." For more information about 32-bit network independent APIs, search your on-line Presentation Manager Programming Reference.
- **Direct printing.** If you must bypass the spooler, you can obtain the names of physical devices by requesting from `SpEnumQueue`, `f1Type SPL_PR_QUEUED_DEVICE`. This `f1Type` lists the physical devices associated with printer objects and identifies them by their logical names, such as "Printer1". Do not request `f1Type SPL_PR_DIRECT_DEVICE`, which in OS/2 1.3 returned devices that have no associated queues. The OS/2 2.0 shell does not permit the user to define a nonqueued print device. It is not recommended that your application define one, either.

It is also possible to print directly to the spooler with a `PM_Q_RAW` job. In the *OS/2 Programming Guide*, see "Submitting a Direct Presentation Manager Print Job" and "Submitting a Print Job Directly to the Spooler" for details.



32-Bit OS/2

DOS Application Printing: Understanding the Differences Under OS/2



Frank J. Schroeder

by Frank J. Schroeder

Many DOS applications can send information to a printer; they implement this feature through a variety of different interfaces. OS/2 provides support for these interfaces, allowing DOS applications to print when running under OS/2. When printing, DOS applications sometimes behave differently under OS/2 than when running under DOS. This article describes the interfaces DOS applications use to send data to a printer, some of the differences encountered when printing under OS/2, and changes made to OS/2 during development that enable DOS applications to print. Some of these changes resulted in options that can be customized to an application. This article will show you how to recognize these differences, apply its recommendations, and gain an appreciation of what has been implemented in OS/2 to maintain DOS compatibility and provide an OS/2 version able to stand up to the claims of being "a better DOS than DOS."

INTRODUCTION

Before describing these behavioral differences and recommendations, this article presents a review of how printing is accomplished from DOS applications. With this knowledge, it is easier to see behavioral differences between DOS and OS/2 as well as how new features in OS/2 2.0 allow a DOS application to print. Some of these new features require tailoring by the user; a proper setting for one DOS application may be improper for the next.

DOS applications send information to be printed using several software interfaces. They are:

- Direct manipulation of parallel port hardware
- BIOS Interrupt 17h
- DOS Interrupt 21h.

The two most popular interfaces for printing are DOS interrupt 21h and BIOS interrupt 17h.

DIRECT ACCESS TO PARALLEL PORT HARDWARE

I do not recommend the direct access to hardware printing approach, as it limits an operating system's ability to coordinate application access to hardware. Problems found in hardware can be worked around by operating system software. Well-written applications are hardware independent; applications should not require change as hardware changes. Very few DOS applications interface directly to the parallel port hardware to print, as shown in Figure 1.

Intermixed printer output occurs in OS/2 1.x if an OS/2 application is printing in the background and a DOS application is printing in the foreground by direct access to hardware. The Intel 80286 processor cannot provide port trapping, which would prevent intermixed output, as shown in Figure 2.

Any DOS application accessing the parallel ports directly in OS/2 2.0 is trapped by the virtual parallel port device driver (VLPT). VLPT asks the physical parallel port device driver if the DOS application may access the hardware directly, through the Inter-device Driver Communication (IDC). If permission is granted, the DOS application is allowed

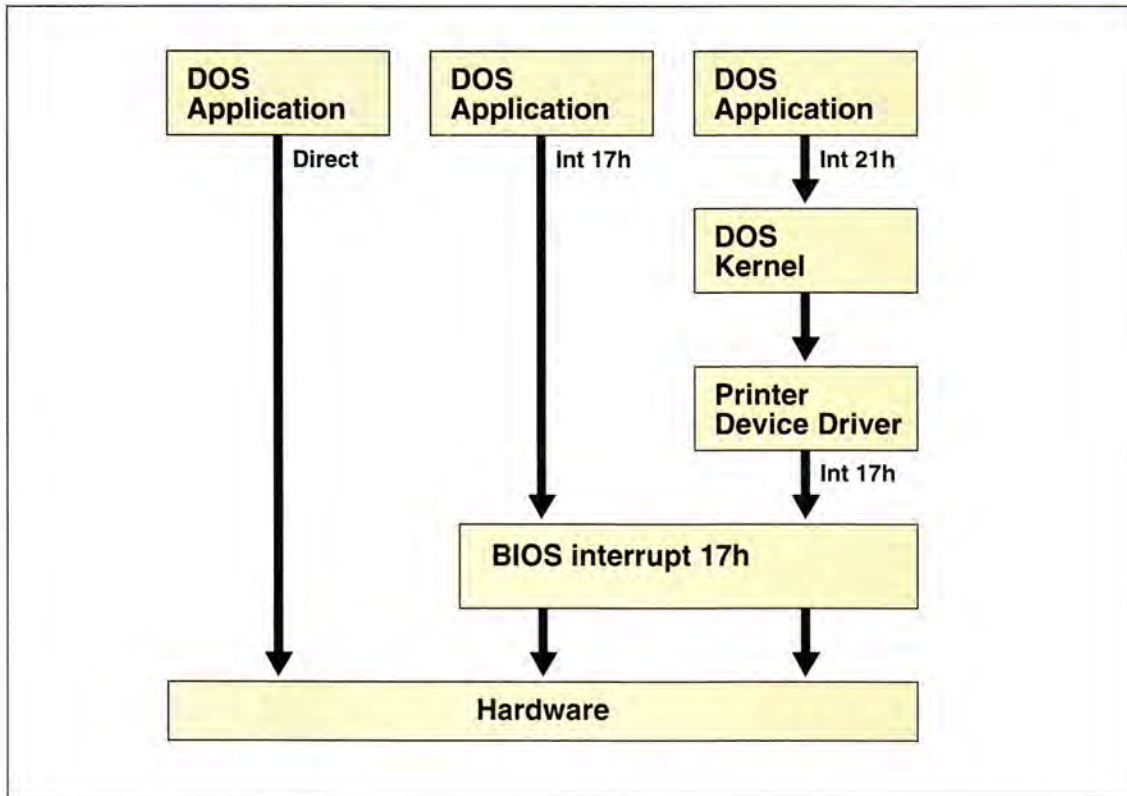


Figure 1: DOS 5.0 printer subsystem

exclusive use of the parallel port hardware until it terminates. Printing from other OS/2 applications can continue to spool but will not be permitted to print until exclusive access is released, as shown in Figure 3.

Most DOS applications do not access hardware directly. For those that do, OS/2 2.0 coordinates this access with other parallel port hardware accesses to prevent intermixed output.

BIOS INTERRUPT 17H

Basic Input/Output System (BIOS) is a set of functions in ROM that perform low-level interfacing with the hardware. DOS uses BIOS interrupt 17h to send data to printers attached to parallel ports, as shown in Figure 1. If a DOS application issues interrupt 21h functions 05h or 40h, they are converted by DOS into BIOS interrupt 17h requests.

In OS/2 1.x versions, there is only one DOS session. The physical parallel port device driver intercepts all requests to interrupt 17h and coordinates DOS application printing with

printing by other applications, as shown in Figure 2. A disabled spooler in OS/2 can result in intermixed printer output.

OS/2 2.0 allows up to 240 DOS sessions. A new component, the virtual device driver, keeps track of DOS applications as they attempt to manipulate hardware or issue BIOS interrupt calls. VLPT handles DOS application accesses to interrupt 17h, buffering these single character requests and passing buffers through the OS/2 kernel to the parallel port device driver for printing, as shown in Figure 3.

Most DOS applications print to interrupt 17h instead of interrupt 21h. Interrupt 17h is a very simple interface which performs better because it is specific for printing and does not require involvement by the DOS operating system. In any case, the DOS operating system passes interrupt 21h print requests to BIOS interrupt 17h. Developers, in an attempt to achieve better performance, have chosen to bypass interrupt 21h and code directly to interrupt 17h. The design of interrupt 17h, which does not include open or close requests to bracket the print request, has led to spooling problems. These issues had to be overcome so

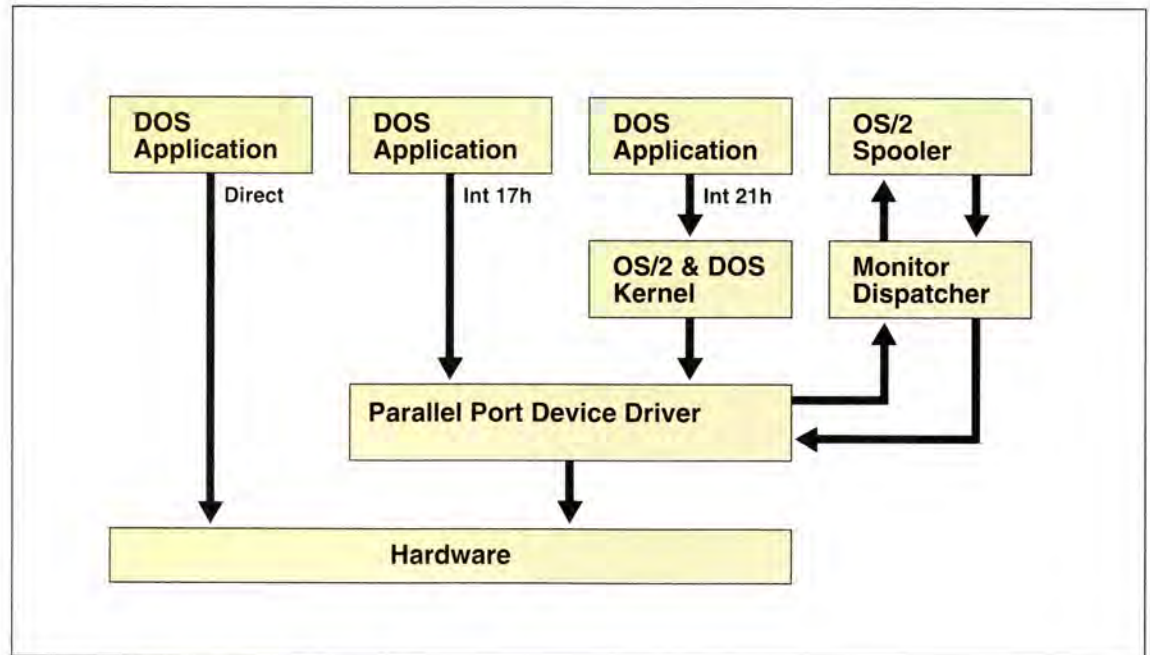


Figure 2: DOS application printing in OS/2 1.x

Most of the different behaviors occur in the interaction between DOS applications and the OS/2 spooler.

these DOS applications could print in a DOS session.

DOS INTERRUPT 21H

DOS applications pass information to the operating system to be printed using Function 05h (Printer Output) or Function 40h (Write to File or Device) of the DOS interrupt 21h interface.

DOS utilizes standard handles, devices opened by the operating system and available without an explicit open request. Similarly, when an application is ready to terminate, it need not issue an explicit close request. One of the five DOS standard handles corresponds to the standard printer, the first parallel port in a system.

When a DOS application is started, the operating system generates an open request on behalf of the application. After receiving a print request from the user, the application can issue multiple interrupt 21h function 05h requests to send data, character by character, to the printer. When the application terminates, the operating system generates the corresponding request to close the data stream.

Interrupt 21h function 40h works a little differently. The DOS application must first

issue an explicit open request (function 3Dh) to open the data stream before issuing the function 40h, which sends data to the printer. This process gives the application more control over which system resources it uses (e.g., file system handles) and when it uses them. Once the DOS application has completed transmitting the data to be printed, it must issue an explicit close request (function 3Eh) to close the data stream.

In OS/2, interrupt 21h requests are converted by the DOS kernel into OS/2 kernel requests, which are then passed to the parallel port device driver for processing, as shown in Figures 2 and 3. Some DOS applications use interrupt 21h function 40h to print; a few of the older ones use interrupt 21h function 05h. Subtle differences in how DOS applications are coded to perform the printing function can have dramatic effects in the multitasking environment of OS/2 when the spooler is enabled.

DOS APPLICATION PRINTING DIFFERENCES

With an understanding of the various printing methods DOS applications use and how DOS and OS/2 handle these methods, we can now begin to understand some of the different DOS

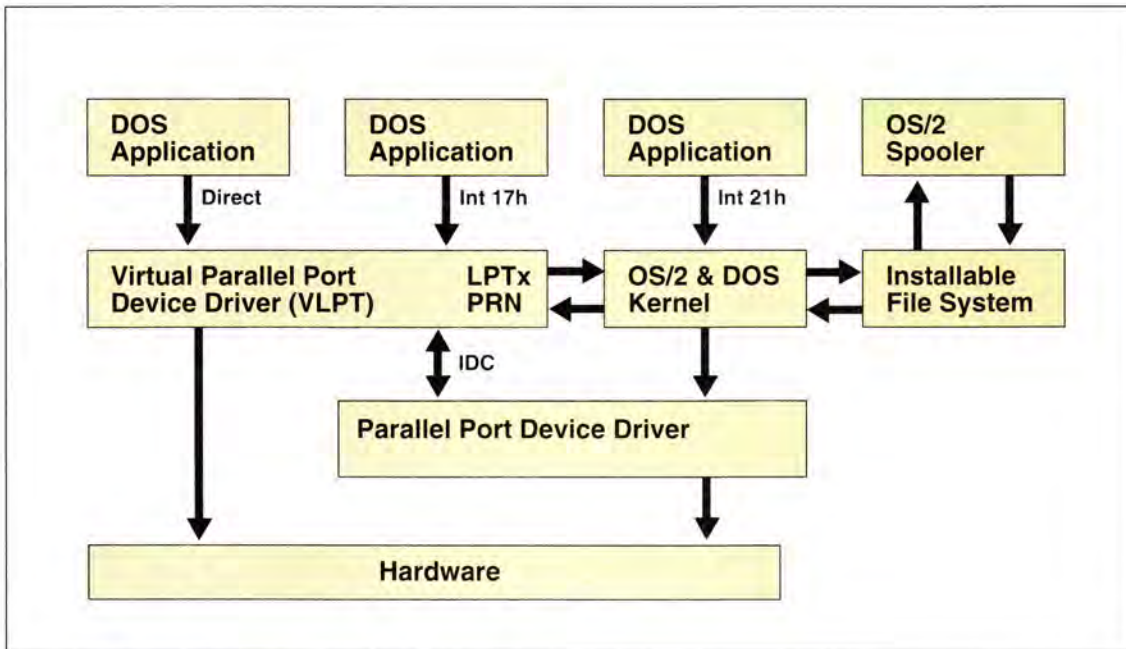


Figure 3: DOS application printing in OS/2 2.0

application printing behaviors users may encounter. Most users do not know or care which method DOS applications use to print. They are interested in hard-copy output; if that output isn't printed properly and quickly, users are upset—and rightly so.

One significant difference between DOS and OS/2 is their single-tasking versus multitasking capabilities. With a single-tasking operating system such as DOS, the task has ownership of all devices and can access them at will. OS/2 must control multiple tasks attempting to access the same hardware and must prevent collisions that cause applications to hang or produce other unexpected errors. The OS/2 spooler handles printer device contention. If it is disabled, multiple applications attempting to print simultaneously can cause intermixed output.

These are some of the differences encountered when DOS applications attempt to print from within a DOS session of OS/2:

- Instantaneous printing
- BIOS Interrupt 17h printing
- Interrupt 17h Terminate-and-Stay Resident (TSR) DOS programs

- Open and close requests with no write request (ghost jobs)
- Open request at initialization and close request at termination
- Open and close request for each application buffer printed
- DOS applications accessing hardware directly.

The next section contains a description of each difference followed by some insight into changes made to OS/2 that allow DOS applications to print. Interacting with the spooler does not necessarily correspond to how DOS applications print. Most different behaviors occur in the interaction between DOS applications and the OS/2 spooler.

INSTANTANEOUS PRINTING

Behavioral Difference

One of the first things a user notices when printing from a DOS application under OS/2 is that the printer does not begin printing as quickly as when running the same application under DOS. This slower response time sometimes translates into a negative perception of OS/2.



Recommendation

One important difference between DOS and OS/2 is the inclusion of a printer spooler. (Compare Figure 1 to Figures 2 and 3.) If the spooler is enabled, the entire print job is sent by the DOS application to the spooler, which must write it to a temporary spool file before it can be sent to the printer. The spooler must behave this way because it is managing serially reusable devices and cannot assign a printer to an application while the application is busy building the file to print. The spooler tries to achieve maximum utilization of the devices it manages; another application may have a small job to print, but cannot do so if the spooler assigned the device to the first application, even if the first application leaves the printer idle while building the output.

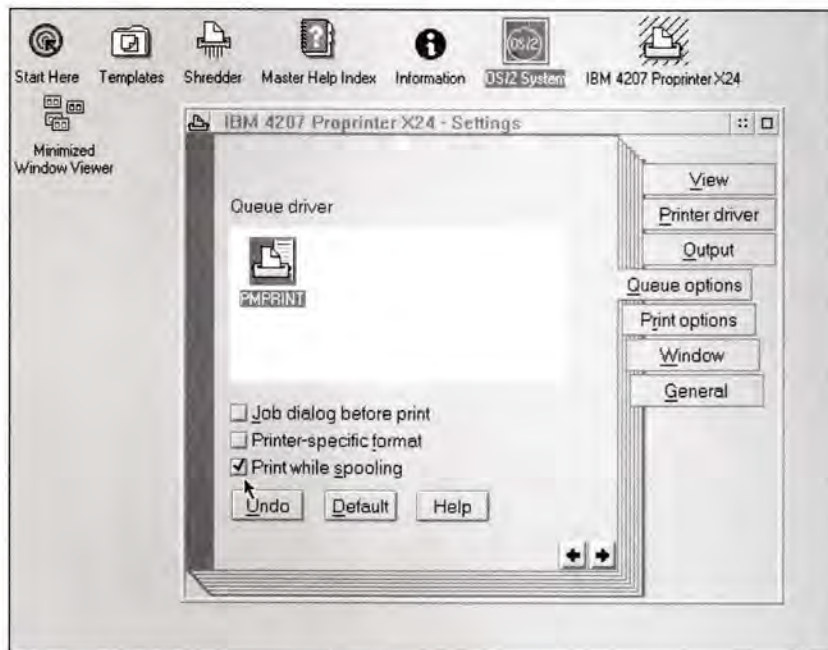


Figure 4: OS/2 2.0 Print while spooling queue option

The spooler is necessary in OS/2 unless the user performs the serialization of accesses to the printer. The spooler can be disabled if a user wants to print from only one application at a time; in this case, printing begins instantaneously.

In OS/2 1.x, spooler design was based on the character monitor mechanism. In OS/2 2.0, it has been redesigned as an installable file system, which improves performance. The

monitor dispatcher has been rewritten using 32-bit instructions, and the parallel port device driver monitor interface has been made wider, improving character monitor throughput and reducing the amount of time before printing begins.

In OS/2 2.0, a new queue option allows printing to commence before spooling is completed, as shown in Figure 4. By selecting the **Print while spooling** checkbox, the user can toggle this option on (default) or off. Print while spooling saves time by starting large print files before the entire file has spooled. A side effect may occur with certain applications. If an application does not issue a close printer request after sending data to print, the next job in the queue cannot begin printing until the close request is received. Users may need to take overt action, such as pressing a special key sequence (Ctrl-Alt-Print Screen in the DOS session) or terminating the application to generate the close. The Print while spooling option is not recommended on print servers.

BIOS INTERRUPT 17H PRINTING

Behavioral Difference

A unique characteristic interrupt 21h has over BIOS interrupt 17h is the ability to identify the beginning and end of the data stream. Each data stream corresponds to one print request and one spool file. When the physical parallel port device driver in OS/2 1.x or the virtual parallel port device driver in OS/2 2.0 sees the first interrupt 17h print request in a DOS session, it recognizes the start of the data stream and informs the spooler to open a spool file. Interrupt 17h print requests can be buffered in the device driver until the buffer becomes full, at which time the buffer can be written into the spool file. Of primary concern is how the device driver can detect the end of the print request so that the data stream can be closed and the spooler can send the request to be printed. In this case, the user sees that the DOS application has "printed," but no hard copy has been generated.

Recommendation

Several methods have been built into OS/2 to start interrupt 17h initiated print requests:

- Application termination
- DOS session termination
- Ctrl-Alt-Print Screen key sequence
- DOS Timeout (OS/2 1.3) or `PRINT_TIMEOUT` DOS Setting (OS/2 2.0)

Users notice the most obvious method when they terminate the DOS application and the job starts printing. The device driver receives notification of application termination, writes any remaining characters in its buffer to the spooler, and closes the spool file. In OS/2 2.0, termination of the DOS session generates the same behavior as terminating the application.

The key sequence Ctrl-Alt-Print Screen, added in OS/2 1.x, causes any remaining data to be written and the spool file closed. This approach requires that the user knows the key sequence, performs the sequence once the DOS application has finished sending the print job, and presses the keys at the appropriate time, since doing so prematurely splits the print job into two spool files. With this option, the user still has to perform an action that is not necessary when printing from the same application under DOS.

A more transparent method called DOS Timeout was added to the OS/2 1.3 Print Manager setup menu to automatically flush the final buffer and close the spool file when interrupt 17h print requests stop occurring. The DOS setting option `PRINT_TIMEOUT` in OS/2 2.0 performs the same function, as shown in Figures 5 and 6.

Following is an example of a setting that must be tailored to the application. Some DOS applications issue interrupt 17h requests to print and generate only some of the print data before issuing more requests and repeating the process. If a DOS application prints in spurts and a very small timeout value is specified, the timeout may expire when the application is generating more print data. The print job may then be split into two spool files. Conversely, if the user specifies an inordinately large timeout value, he or she may wait unnecessarily before the final buffer is sent to the spooler, the spool file is closed, and the job begins to print. (Subtleties like these exist even between DOS

applications using the same interface to print.)

Once the end of a data stream is determined, the start of the next job can be determined and the process can repeat itself for the next print job within that DOS session.

INTERRUPT 17H TERMINATE-AND-STAY RESIDENT (TSR) DOS PROGRAMS

Behavioral Difference

Since DOS maps interrupt 21h print requests to interrupt 17h, DOS application developers have developed TSR applications that replace the BIOS interrupt 17h routine in the interrupt vector table with an entry point into their TSR application. When print requests occur, the TSR gains control and performs its operation before printing by directly accessing the parallel port hardware or passing the request on to the BIOS interrupt 17h routine. The user wonders why the DOS application (TSR) doesn't see print requests under OS/2.

Some settings need to be tailored to the application.

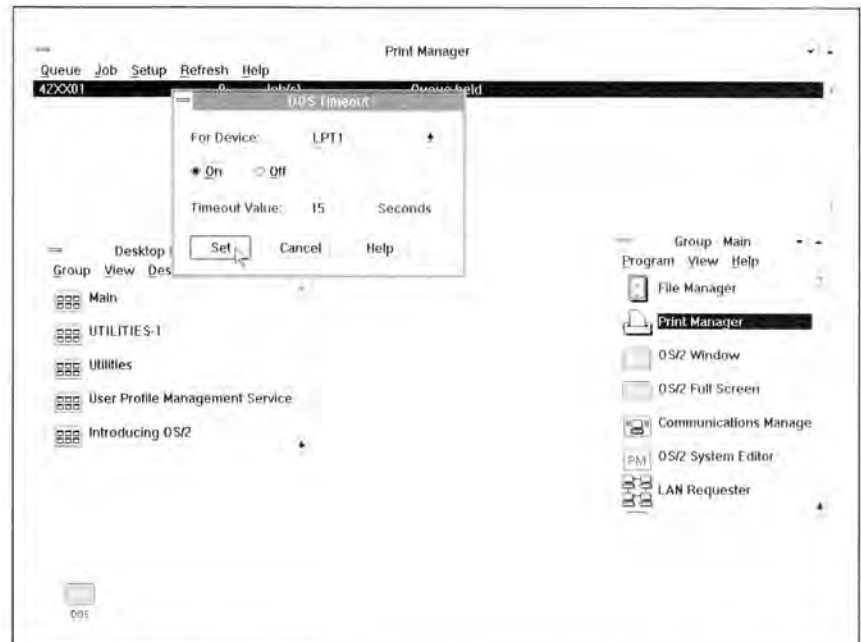


Figure 5: DOS timeout option in OS/2 1.3

Recommendation

When a DOS application issues interrupt 21h print requests in OS/2, the DOS kernel



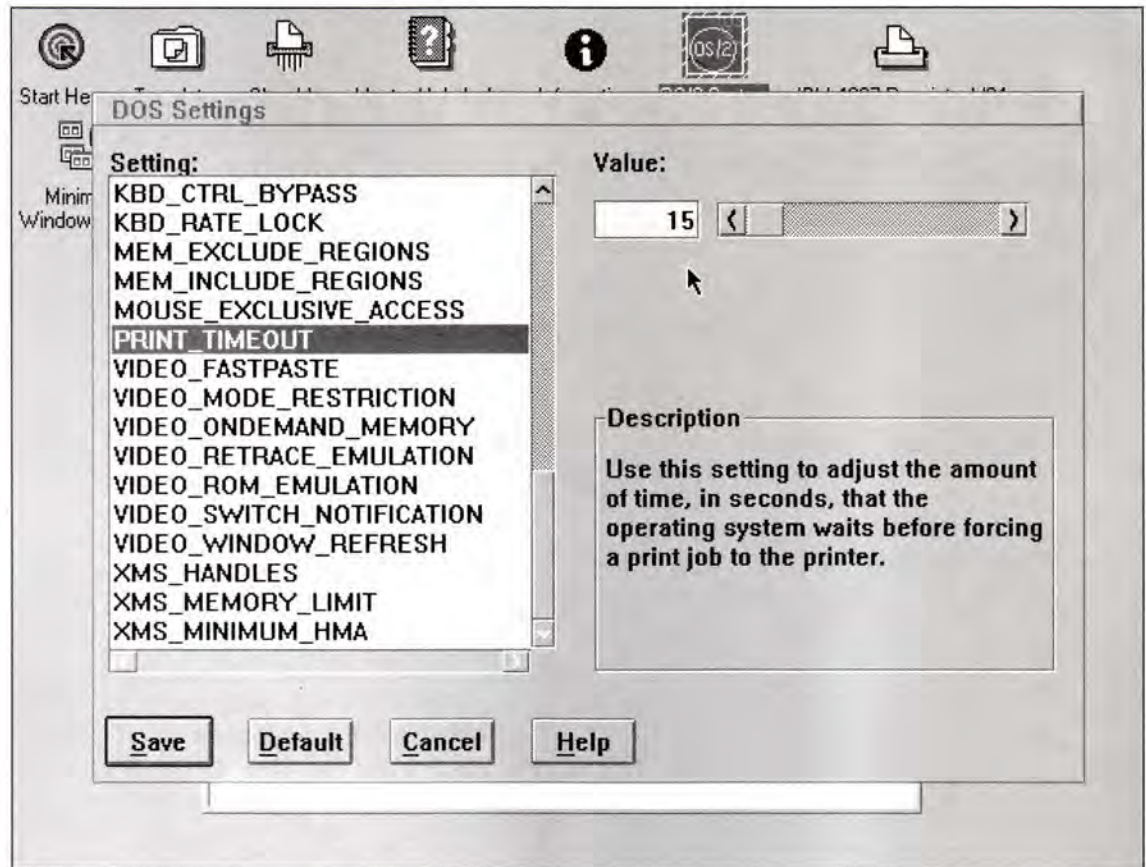


Figure 6: PRINT_TIMEOUT DOS setting in OS/2 2.0

receives control and sends buffers to the parallel port device driver to print instead of converting them to interrupt 17h requests, as under DOS. This OS/2 performance enhancement prevents DOS TSRs from being able to perform their function.

OS/2 2.0 has shipped a DOS device driver called LPTDD.SYS which converts interrupt 21h requests to interrupt 17h requests so the TSR class of DOS applications will work properly. This device driver is installed into the \OS2\MDOS subdirectory but is not loaded by default.

A user can load the DOS device driver using the `DEVICE=C:\OS2\MDOS\LPTDD.SYS` statement in `config.sys` so it applies to all DOS sessions or add `C:\OS2\MDOS\LPTDD.SYS` to the `DOS_DEVICE` DOS setting so it applies to one particular DOS session, as shown in Figure 7.

OPEN AND CLOSE REQUESTS WITH NO WRITE REQUEST

Behavioral Difference

When DOS applications are loaded to run, OS/2, like DOS, opens the standard printer handle. When DOS applications terminate, OS/2 closes this handle. The spooler generates a temporary spool file when it receives a printer open request and closes the file with a corresponding close request. In beta testing, if a user looked at the spool queue (in the Print Manager of OS/2 1.x or the Print Object of OS/2 2.0), a job used to appear and then disappear but nothing was printed. This behavior gave the impression of a lost spooler output file, commonly dubbed a "ghost file," when, in reality, nothing was ever printed by the application using DOS interrupt 21h function 05h.

A similar problem occurred if code page switching was enabled or the form-feed

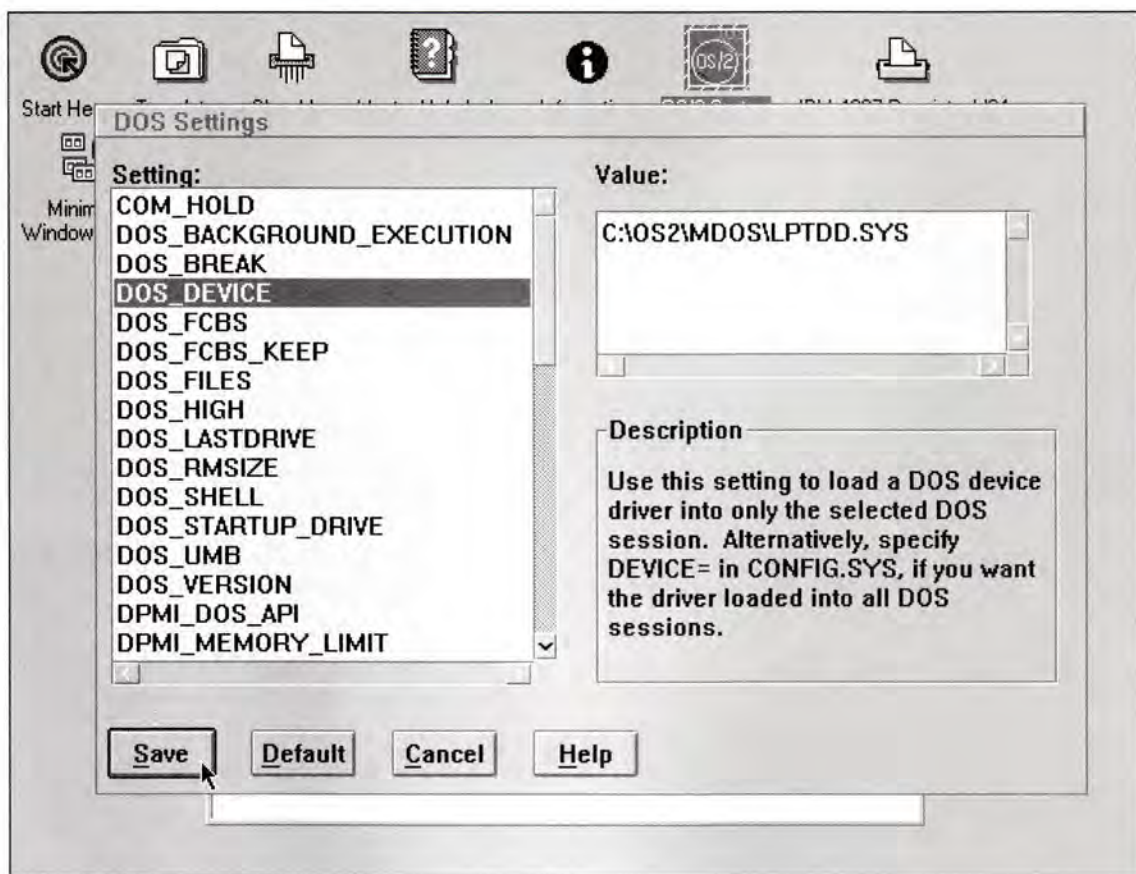


Figure 7: DOS_DEVICE DOS setting in OS/2 2.0

control was set within the printer properties dialogue of the printer driver. If the form-feed option was activated, a blank sheet of paper passed through the printer.

Recommendation

In this case, the spooler was modified so it would not alert the spool queue of the print job until data was received and written to a spool file. This hid the open and close requests. The spooler was optimized to discard empty spool files or files that contained only printer code page downloads, so users should no longer see ghost files in the spool queue.

OPEN REQUEST AT INITIALIZATION AND CLOSE REQUEST AT TERMINATION

Behavioral Difference

Some DOS applications open a file handle to the printer using interrupt 21h function 3Dh

when the application does its initialization code and close it using interrupt 21h function 3Eh at termination processing. If a user requests printer output from within a DOS application written with this algorithm, the print job is spooled but remains in the "spooling" state, as the application doesn't close the handle until it is terminated. The spooler needs to receive the close request as a signal that the application is finished sending data to the printer so printing can commence.

Recommendation

The DOS device driver LPTDD.SYS can convert interrupt 21h print requests into interrupt 17h print requests. One of the methods described under *BIOS Interrupt 17 Printing* can be used to close the spool file. The spooler will then automatically initiate printing.



OPEN AND CLOSE REQUEST FOR EACH APPLICATION BUFFER PRINTED

Behavioral Difference

A few DOS applications open and close the printer handle using interrupt 21h function 3Dh and 3Eh for each buffer of a print job being sent to a printer. Although this behavior is not a problem in DOS, under OS/2 it causes multiple spool files to be generated for one print job. If a user holds the spool queue and prints with an application written this way, many files appear in the spool queue and give the impression that he or she has requested that the DOS application print the job once for each spool file in the queue. In reality, only one job is in the queue but it has been split into many pieces.

Recommendation

This is usually not a concern; however, it does clutter up the spool queue and add unnecessary overhead to the system. OS/2 has not done any work to detect this situation or collapse these files into one spool file, as this problem has not been widespread. The Browse Spool File option in OS/2 2.0 will not provide the desired results, as only partial output can be viewed.

DOS APPLICATIONS ACCESSING HARDWARE DIRECTLY

Behavioral Difference

An operating system can do little to prevent DOS applications running under DOS or OS/2 1.x from directly accessing PC hardware. An application that directly manipulates hardware can compromise system integrity, especially if the hardware is left in an altered state.

A user may find that one DOS program printed, but print jobs now spool but won't print. The LPTx queue appears hung, the printer is ready, and other queues print with no problems. The user cannot detect what could be wrong.

Recommendation

Using the port-trapping feature of the Intel 80386 and 80486 processors, the virtual parallel port device driver of OS/2 2.0 traps the attempts to manipulate the parallel port hardware and coordinates this hardware access with the operating system's hardware access. When a DOS application terminates, exclusive control is returned to the operating system, and the port is reset to its default state.

There is a limitation when DOS applications in OS/2 access hardware directly. While the DOS application is touching the hardware, no other applications, including other instances of the same application, can touch the same hardware. The spooler is prevented from printing by the parallel port device driver until the DOS application terminates. Several classes of DOS applications fit into this category, including file transfer programs and software security devices attached to the parallel port.

CONCLUSION

This article has described the methods DOS applications use to print, how OS/2 tries to accommodate these applications, and some customizations available in OS/2 that enable DOS applications to peacefully coexist with OS/2 and Windows applications. This article has also explained some of the behind the scenes architectural decisions that enable DOS applications to perform better under OS/2.

Frank J. Schroeder, IBM Entry Systems Division, 1000 N.W. 51st Street, Boca Raton, Fla. 33429. Schroeder is a staff programmer in the OS/2 Device Drivers and MVDm department. He joined IBM in 1984 and is currently working on performance and functional enhancements to the OS/2 Printer Subsystem. He holds a B.T. in computer science from Rochester Institute of Technology in New York, and an M.B.A. in business administration from the University of Miami, Florida.

Class Objects in SOM



by Nurcan Coskun and Roger Sessions

This article continues the discussion of SOM that began in the Winter 1992 issue of IBM Personal Systems Developer. This article examines the class object, an advanced programming feature of SOM. At run time, a class object can retrieve information about a class, such as the class name and supported methods, and can also instantiate objects of a given class. We will give an overview of class objects and discuss metaclass, or the class of a class object. We will also demonstrate two techniques for changing a metaclass.

object-oriented programming, such as developing classes and instantiating objects, deriving new classes through inheritance, and resolving methods with polymorphism. We also covered the primary goals of SOM: multilanguage support for procedural and object-oriented languages, upwardly compatible binaries, and object-oriented extensions to non-object-oriented languages.

With our previous article as a base, this article will explore class objects in SOM. For full information on using SOM, see the SOM guide and reference (IBM, SOM, 1992).



Roger Sessions

INTRODUCTION

In our last article (Sessions and Coskun 1992, 107-119), we gave a general introduction to the System Object Model, or SOM. Introduced by IBM with OS/2 2.0, SOM is a new methodology for creating object-oriented class libraries. We discussed the basic concepts of

GOALS OF SOM

SOM is a language-neutral mechanism for developing object-oriented class libraries, consisting of a run-time repository for class information and a series of language bindings. SOM's structure is diagrammed in Figure 1.



Nurcan Coskun

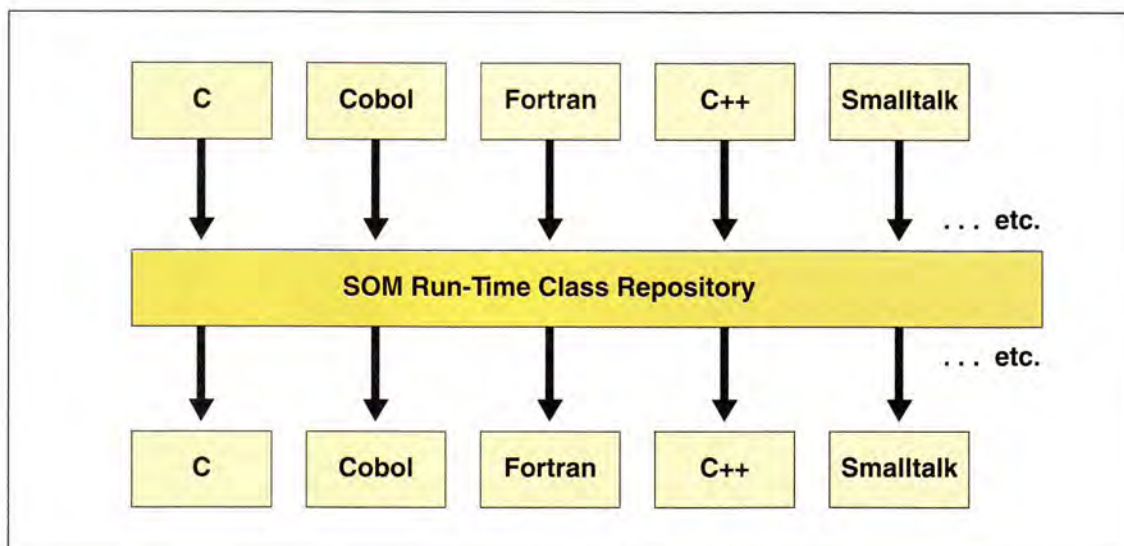


Figure 1: Pictorial representation of the System Object Model (SOM)



The box in Figure 1 with C pointing to the Run-Time Repository represents a binding that allows classes to be defined in C and added to the SOM run-time repository. The connection between the run-time repository and C is a binding that allows C to use classes from the repository regardless of original language. "Using a class" means either instantiating and invoking methods on objects of a particular class or deriving new classes using that class as a base.

OS/2 2.0 currently supports only C bindings; other languages in Figure 1 are for illustration only. While we are working with a wide variety of language suppliers to increase the number of supported languages, this is neither a complete nor committed list of languages under consideration.

As we discussed in our first article, C bindings allow object-oriented programs to be developed much as they are in C++, with the difference that classes in SOM can be used by a variety of languages as new language bindings become available. SOM improves on other class library weaknesses in C++; for an introduction to C++ and library issues, see *Class Construction in C and C++* (Sessions 1992).

CLASSES IN SOM

Let's consider a set of classes we first looked at in the Winter issue, starting with the `Student` class. An object of type `Student` knows how to set up its ID and name, how to return its ID, and how to print itself. The class definition file for `Student` is shown in Figure 2.

Figure 3 shows the class implementation file for `Student`. Most of this file is produced automatically by the SOM compiler; the lines the class implementor needs to write are shown in bold face.

Given the definition of `Student`, we can derive a new class called `GraduateStudent`, which has additional information on thesis and degree. The class definition file for `GraduateStudent` is shown in Figure 4, and the class implementation file is shown in Figure 5.

With these two classes defined and implemented, we can write client code to instantiate and manipulate `Students` and

```
include <somobj.sc>
class:
    Student;
parent:
    SOMObject;
data:
    char id[16];      /* student id */
    char name[32];    /* student name */
methods:
    override somInit;
    void setUpStudent(char *id, char *name);
    void printStudentInfo();
    char *getStudentType();
    char *getStudentId();
```

Figure 2: `Student` class definition

`GraduateStudents`. The program in Figure 6 instantiates two `Students` and two `GraduateStudents`, initializes the objects with data, and asks the objects to print themselves.

CLASS OBJECTS

Every class with at least one instantiated object has exactly one associated class object. Each instantiated object maintains a pointer to its class object, which it can return via `somGetClass`. This method is inherited from `SOMObject`, a class from which every class is derived either directly or indirectly. In the program in Figure 6, `student1` and `student2` are both objects of class `Student`, and therefore both share the same class object. Similarly, `grad1` and `grad2` are both objects of class `GraduateStudent` and share a second, different class object. The relationship between objects and their class object is shown in Figure 7.

Another way to access a class object is through the SOM defined macro `_classObj`, which returns a pointer to the appropriate class object, or to zero if none exists. For `Student`, the macro is `_Student`. SOM instantiates a class object the first time an object of its controlled class is instantiated. The `_Student` will therefore return 0 if and only if no `Student` objects have yet been instantiated.

The class object can tell you many things about a class, including its size, its parent, whether it is derived from some other class, and whether



```

#define Student_Class_Source
#include "student.ih"

void somInit(Student *somSelf)
{
    StudentData *somThis = StudentGetData(somSelf);
    parent_somInit(somSelf);
    _id[0] = _name[0] = '\0';
}

void setUpStudent(Student *somSelf,
    char *id, char *name)
{
    StudentData *somThis = StudentGetData(somSelf);
    strcpy(_id, id);
    strcpy(_name, name);
}

void printStudentInfo(Student *somSelf)
{
    StudentData *somThis = StudentGetData(somSelf);
    printf("\n   Id       : %s \n", _id);
    printf("   Name      : %s \n", _name);
    printf("   Type       : %s \n", _getStudentType(somSelf));
}

char * getStudentType(Student *somSelf)
{
    StudentData *somThis = StudentGetData(somSelf);
    static char *type = "student";
    return (type);
}

char * getStudentId(Student *somSelf)
{
    StudentData *somThis = StudentGetData(somSelf);
    return (_id);
}

```

Figure 3: Student class implementation (programmer added lines shown in boldface)

it supports a specific method. The class of a class object is either `SOMClass` or some class derived from it. A variety of methods defined for `SOMClass` are documented in the class definition file for `SOMClass`, `somcls.sc`, and the `SOM` documentation.

One `SOMClass` method is `somGetName()`, which returns a pointer to a character string containing the name of the class for which you have indicated the class object. For the `Student` class object, this string would be "Student."

Another interesting `SOMClass` method is `somNew()`, which returns a newly instantiated

object of the appropriate class. For the `Student` class object, it would return a newly instantiated object of the `Student` class. Although `somNew()` is always invoked to instantiate new objects, many programmers are unaware of this because they use a macro wrapper of the form `classnameNew` (for example, `StudentNew()`). The wrapper first makes sure the appropriate class object exists and then makes the instantiation request via `somNew()`.

In most cases, we would instantiate objects using the class instantiation macro, as with `StudentNew()`. There are some cases, however, in which the class object but not the class



```
include "student.sc"
class:
  GraduateStudent;
parent:
  Student, local;
data:
  char thesis[128]; /* thesis title */
  char degree[16]; /* graduate degree type */
methods:
  override somInit;
  override printStudentInfo;
  override getStudentType;
  void setUpGraduateStudent(char *id, char *name,
```

Figure 4: GraduateStudent class definition file

instantiation macro must be accessed, such as a function that accepts a pointer to any object and returns another newly instantiated object of the same class. Figure 8 modifies the program in Figure 6 to make use of the `Student` and `GraduateStudent` class objects and the `somGetClass()` and `somNew()` methods.

Several of the `SOMClass` methods have passthrough methods defined in `SOMObject`. In these cases, the code for the passthrough methods simply gets the object's class object and passes through the call using the appropriate `SOMClass` method. For example, the `SOMObject` method `somGetClassName()` passes through to the `SOMClass` method `somGetName()`. This article uses either the `SOMClass` methods or the `SOMObject` passthrough, depending on the point to be made.

THE CLASS OF THE CLASS OBJECT

Every object in SOM is associated with a class. The class of `student` is `Student`:

```
#include "student.h"

main()
{
  Student *student = StudentNew();
  printf("student object is of <%s> class\n",
        _somGetClassName(student));
}
```

which gives the output:

student object is of <Student> class

Class objects are also associated with a class. Because `SOMClass` is, like all classes, derived from `SOMObject`, it therefore supports all `SOMObject` methods. We can, for example, verify the class of the `Student` class object:

```
#include "student.h"

main()
{
  Student *student = StudentNew();
  SOMClass *studentClassObject;

  studentClassObject = _somGetClass(student);
  printf("student object is of <%s> class\n",
        _somGetClassName(studentClassObject));
}
```

giving the output:

student class object is of <SOMClass> class

We use "metaclass" to describe the class of a class object. For example, the class of the student's class object is `SOMClass`; thus the metaclass of `Student` is `SOMClass`. Class and metaclass should not be confused; the class of `student` is `Student`, while the metaclass of `student` is `SOMClass`.

We can change the metaclass of a given class by a two stage process:

- Derive a new class from `SOMClass`, either directly or indirectly


```

#define GraduateStudent_Class_Source
#include "graduate.ih"

void somInit(GraduateStudent *somSelf)
{
    GraduateStudentData *somThis =
        GraduateStudentGetData(somSelf);
    parent_somInit(somSelf);
    _thesis[0] = _degree[0] = '\0';
}

void printStudentInfo(
    GraduateStudent *somSelf)
{
    GraduateStudentData *somThis =
        GraduateStudentGetData(somSelf);
    parent_printStudentInfo(somSelf);
    printf("    Thesis    : %s \n", _thesis);
    printf("    Degree    : %s \n", _degree);
}

char * getStudentType(
    GraduateStudent *somSelf)
{
    GraduateStudentData *somThis =
        GraduateStudentGetData(somSelf);
    static char *type = "Graduate";
    return (type);
}

void setUpGraduateStudent(
    GraduateStudent *somSelf, char *id, char *name,
    char *thesis, char *degree)
{
    GraduateStudentData *somThis =
        GraduateStudentGetData(somSelf);
    _setUpStudent(somSelf, id, name);
    strcpy(_thesis, thesis);
    strcpy(_degree, degree);
}

```

Figure 5: GraduateStudent class implementation file (programmer added lines shown in boldface)

- Use the metaclass directive to tell the class of its new class object.

There are at least three reasons to change the class of a class object:

- To add methods for initializing new objects (constructor methods)
- To modify how memory is allocated for objects
- To keep track of global information about the class, such as the number of instantiated objects.

Let's go through the steps necessary to change the metaclass of `student` from `SOMClass` to `StudentFactory` (in other words, to change the the class of the `Student` class object from `SOMClass` to `StudentFactory`). Let's say we want `StudentFactory` to keep count of the number of instantiated students, functionality not provided by the standard `SOM` class. The steps are:

- Define `StudentFactory` to be derived from `SOMClass`. This adds one new method, `countObjects()`, to `StudentFactory`.



```

Program:
#include "graduate.h"

main()
{
    Student *student1 = StudentNew();
    Student *student2 = StudentNew();
    GraduateStudent *grad1 = GraduateStudentNew();
    GraduateStudent *grad2 = GraduateStudentNew();

    _setUpStudent(student1,"120455","Joe Doe");
    _setUpStudent(student2,"103606","Mary Smith");
    _setUpGraduateStudent(grad1,"203230","Janet Brown",
                          "Parser Generators", "M.S.");
    _setUpGraduateStudent(grad2,"240900","Mark Black",
                          "Code Optimization", "Ph.D.");

    _printStudentInfo(student1);
    _printStudentInfo(student2);
    _printStudentInfo(grad1);
    _printStudentInfo(grad2);
}

```

```

Output:
  Id      : 120455
  Name    : Joe Doe
  Type    : student

  Id      : 103606
  Name    : Mary Smith
  Type    : student

  Id      : 203230
  Name    : Janet Brown
  Type    : Graduate
  Thesis  : Parser Generators
  Degree  : M.S.

  Id      : 240900
  Name    : Mark Black
  Type    : Graduate
  Thesis  : Code Optimization
  Degree  : Ph.D.

```

Figure 6: Client program for Student and GraduateStudent

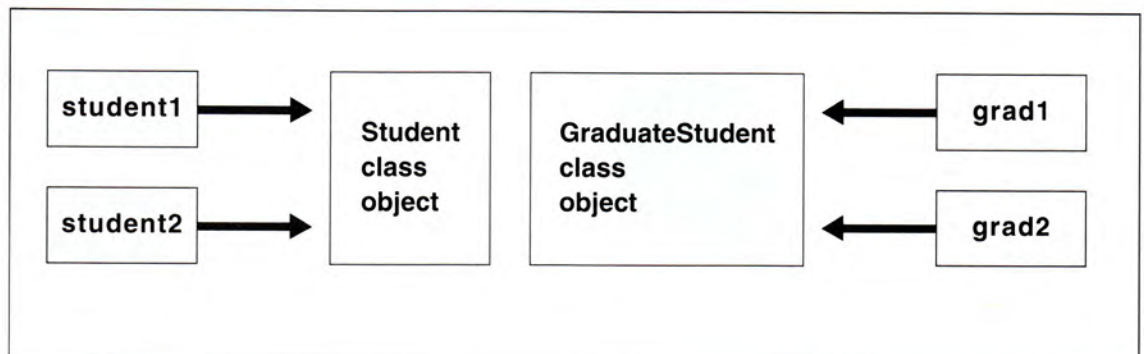


Figure 7: Shared class objects



```
#include "graduate.h"

SOMAny *createObject(SOMAny *object)
{
    return(_somNew(_somGetClass(object)));
}

main()
{
    Student *student1 = StudentNew();
    GraduateStudent *grad1 = GraduateStudentNew();
    Student *student2;
    GraduateStudent *grad2;

    student2 = createObject(student1);
    grad2 = createObject(grad1);

    _setUpStudent(student1, "120455", "Joe Doe");
    _setUpStudent(student2, "103606", "Mary Smith");
    _setUpGraduateStudent(grad1, "203230", "Janet Brown",
                          "Parser Generators", "M.S.");
    _setUpGraduateStudent(grad2, "240900", "Mark Black",
                          "Code Optimization", "Ph.D.");

    _printStudentInfo(student1);
    _printStudentInfo(student2);
    _printStudentInfo(grad1);
    _printStudentInfo(grad2);
}
```

Any number of classes can be associated with a new metaclass.

Figure 8: Client program using class object

- Override the client invoked SOMClass method `somNew()`, which instantiates a new `Student` object.
- Override the `somInit()` method invoked on a newly instantiated class object to initialize the count to zero. The class definition and implementation files for `StudentFactory` are shown in Figure 9.
- Modify the student class definition files as shown in Figure 10. No changes are needed to the class implementation file for `Student`.

You can now use the new `StudentFactory` object to find out how many student objects have been instantiated:

```
#include "student.h"

main()
{
    Student *student1 = StudentNew();
```

```
    Student *student2 = StudentNew();
    SOMClass *studentClassObject;
    _setUpStudent(student1, "120455", "Joe Doe");
    _setUpStudent(student2, "103606", "Mary Smith");

    studentClassObject = _somGetClass(student1);
    printf("Student count: %d \n",
          _countObjects(studentClassObject));
    printf("student1 class object is of <%s> class\n",
          _somGetClassName(studentClassObject));
}
```

with the output:

```
Student count: 2 student1 class object is of
<StudentFactory> class
```

IMPLIED METAClass

Any number of classes can be associated with a new metaclass. For example, instead of `StudentFactory`, we might have used the more



```

Class Definition File:
include <somcls.csc>

class:
    StudentFactory;
parent:
    SOMClass;
data:
    int count;
methods:
    override somInit;
    override somNew;
    int countObjects();

Class Implementation File:
#define StudentFactory_Class_Source
#include "countmet.ih"
void somInit(StudentFactory *somSelf)
{
    StudentFactoryData *somThis = StudentFactoryGetData(somSelf);
    parent_somInit(somSelf);
    _count = 0;
}
SOMAny * somNew(StudentFactory *somSelf)
{
    StudentFactoryData *somThis = StudentFactoryGetData(somSelf);
    _count ++;
    return (parent_somNew(somSelf));
}
int countObjects(StudentFactory *somSelf)
{
    StudentFactoryData *somThis = StudentFactoryGetData(somSelf);
    return (int) _count;
}

```

Figure 9: Class definition and implementation files for StudentFactory

general metaclass name `CountFactory` to track total instantiated objects for many classes, and then used this metaclass whenever we want to keep track of total instantiated objects.

Often, we do not want generality, intending a particular class object to be associated with only one class. For example, if the only change to the metaclass is the addition of new constructor methods applicable to a specific class, we can use a short cut called an implied metaclass to redefine the metaclass. The main advantage of implied metaclass is that the

class definition file for the metaclass can be folded into the class definition file for the class it controls, and similarly for the class implementation.

The student definition and implementation files are shown in Figures 11 and 12, respectively. Two keywords in the class definition define implied metaclass: "classprefix" indicates a prefix used to identify methods associated with the class object. "Class" identifies data members or methods associated with the class object.


```

include <somobj.sc>
include "countmet.sc"

class:
  Student;
metaclass:
  StudentFactory, local;
parent:
  SOMObject;
data:
  char id[16];      /* student id */
  char name[32];   /* student name */
methods:
  override somInit;
  void setUpStudent(char *id, char *name);
  void printStudentInfo();
  char *getStudentType();
  char *getStudentId();

```

Figure 10: Newly modified **Student** class definition files (modified lines are shown in boldface)

```

include <somobj.sc>

class:
  Student, classprefix = StudentClass_;
parent:
  SOMObject;
data:
  char id[16];      /* student id */
  char name[32];   /* student name */
  int count, class;
methods:
  override somInit;
  void setUpStudent(char *id, char *name);
  void printStudentInfo();
  char *getStudentType();
  char *getStudentId();
  override somInit, class;
  override somNew, class;
  int countObjects(), class;

```

Figure 11: **Student** class definition using implied metaclass (changes from original **Student** shown in boldface)



```

#define Student_Class_Source
#include "student.ih"

SOM_Scope void SOMLINK somInit(Student *somSelf)
{
    StudentData *somThis = StudentGetData(somSelf);
    parent_somInit(somSelf);
    _id[0] = _name[0] = '\0';
}
/* Other Student Methods, as before, followed by... */
#undef SOM_CurrentClass
#define SOM_CurrentClass SOMMeta

SOM_Scope void SOMLINK StudentClass_somInit(
    M_Student *somSelf)
{
    M_StudentData *somThis = M_StudentGetData(somSelf);
    parent_somInit(somSelf);
    _count = 0;
}
SOM_Scope SOMAny *SOMLINK StudentClass_somNew(
    M_Student *somSelf)
{
    M_StudentData *somThis = M_StudentGetData(somSelf);
    _count ++;
    return (parent_somNew(somSelf));
}
SOM_Scope int SOMLINK StudentClass_countObjects(
    M_Student *somSelf)
{
    M_StudentData *somThis = M_StudentGetData(somSelf);
    return (int) _count;
}

```

Figure 12: Student class implementation using implied metaclass (programmer added lines shown in boldface)

The output, which had looked like:

```

Student count: 2
student1 class object is of <StudentFactory> class

```

now looks like:

```

Student count: 2
student1 class object is of <M_Student> class

```

The difference in output reflects the system assigned name of the metaclass.

SUMMARY

The class object is an important source of run-time information in SOM. The class object is a fully functional object like any other object in SOM. It has its own class (default `SOMClass`) that can be modified with any of the normal class modification techniques, including inheritance and polymorphism.

The class object is an important programmer resource. The source of run-time class information, it gives you the ability to make



run-time decisions about how objects are manipulated. Because it is a standard SOM object, instantiated from a standard SOM class, there is considerable flexibility to redefine its characteristics as needed.

ACKNOWLEDGMENTS

SOM is the work of many people. Mike Conner developed the idea and implementation and continues to lead the overall design of SOM. Andy Martin designed the SOM Class Interface Language and designed and implemented the class Interface Compiler. Larry Raper implemented many features of the run-time library and ported SOM to OS/2. Larry Loucks provided close technical tracking and was instrumental in focusing our efforts. Other SOM developers include Nurcan Coskun, Scott Danforth, Hari Madduri, Simon Nash, Roger Sessions, and John Wang. The project is managed by Jerry Ronga.

REFERENCES

Sessions, Roger. *Class Construction in C and C++: Object-Oriented Programming Fundamentals*, Englewood Cliffs, N.J.: Prentice Hall, 1992 (IBM Doc. S242-0086).

Sessions, Roger and Nurcan Coskun. "Object-Oriented Programming in OS/2 2.0," *IBM Personal Systems Developer* (Winter 1992): 107-119.

SOM: OS/2 2.0 Technical Library System Object Model Guide and Reference, Englewood Cliffs, N.J.: Prentice Hall, 1992 (IBM Doc. 10G6309).

Roger Sessions, IBM Corp., 11400 Burnet Rd., Austin, Texas, 78758. Sessions is the author of two books, *Reusable Data Structures for C and Class Construction in C and C++: Object-Oriented Programming Fundamentals*, as well as several articles. He is working on object-oriented programming environments and previously worked with high-performance relational databases and object-oriented storage systems. Sessions can be contacted at roger@vnet.ibm.com. He holds a B.A. from Bard College and an M.E.S. in database systems from the University of Pennsylvania.

Nurcan Coskun, IBM Corp., 11400 Burnet Rd., Austin, Texas, 78758. Dr. Coskun's expertise is in integrated programming environments, code generators, incremental compilers, interpreters, language based editors, symbolic debuggers, application frameworks, and language design. She previously worked on the OS/2 Database Manager and is now working on object-oriented programming environments. Dr. Coskun can be contacted at nurcan@ausvm1.vnet.ibm.com. She holds a B.S. in industrial engineering from Middle East Technical University and an M.S. and a Ph.D. in computer science from the University of Missouri, Rolla.



32-Bit OS/2

The OS/2 Workplace Programming Interface

by Mary A. Wright

In OS/2 1.x, the Desktop is a collection of windows or icons representing windows associated with applications. In OS/2 2.0, the Desktop is a collection of objects (or icons) and windows associated with those objects. The Desktop (which is also an object), the objects that appear on the Desktop, and the underlying code supporting these objects constitute the OS/2 Workplace Shell, the default user interface for OS/2 2.0.

easier. However, an object-oriented user interface can be developed with more traditional programming languages and tools.

The OS/2 2.0 Workplace Shell is an example of a user interface developed using object-oriented programming, specifically the IBM System Object Model (SOM). Application developers can easily design applications for the new paradigm. Workplace applications consist of sets of related objects used to perform a set of tasks. In addition to providing a consistent and easy-to-use user interface, Workplace applications offer all the advantages provided by object-oriented programming techniques, such as modularity, reusability, and so on.

This article is an introduction to the Workplace programming interface, which consists of the predefined Workplace classes and functions provided with the OS/2 2.0 Toolkit. Application developers can take advantage of the function provided by the Workplace by deriving their new classes from, or subclassing, the predefined Workplace classes.

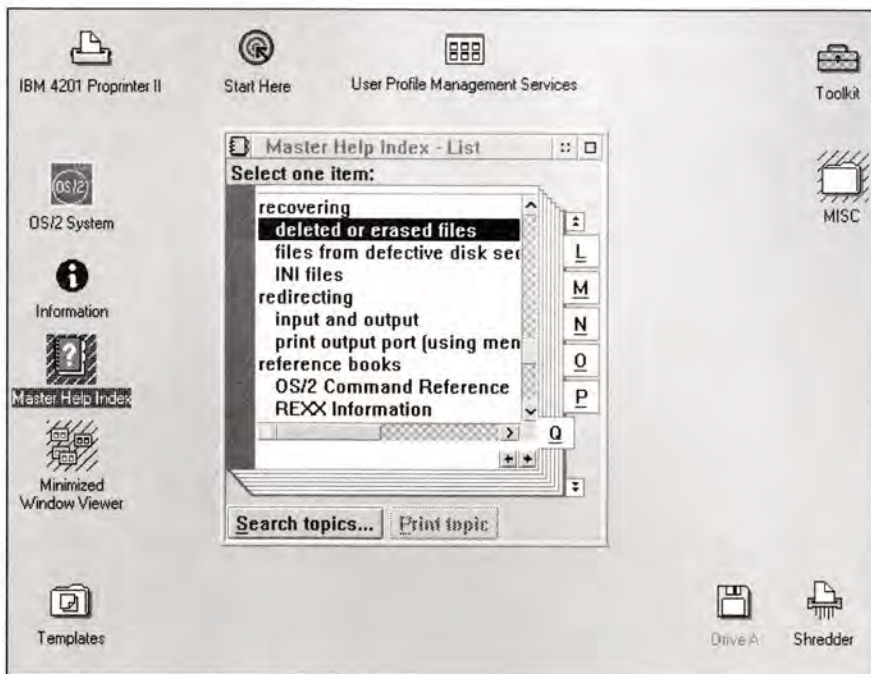


Figure 1: Workplace Shell desktop

While object-oriented user interfaces share some concepts with object-oriented programming, user objects may not necessarily correspond to software objects. Object-oriented programming can make the development of an object-oriented user interface

THE WORKPLACE SHELL

Prior to 1991, Common User Access (CUA) guidelines defined the user interface for application-oriented environments. In application-oriented environments such as OS/2 1.x, users perform tasks by starting an application. They start an application by selecting an installed program from a list displayed in a window or by entering the program name at a command-line prompt. For example, users can start an editor to create and edit a file, start a spreadsheet application to create and update a spreadsheet, or send a file to a printer.



In 1991, CUA guidelines defined the user interface for object-oriented environments. In object-oriented environments such as OS/2 2.0, users perform tasks by manipulating on-screen objects. Users do not start an application to perform a task. Instead, they select an action or drag the object and drop it on another object to perform a task. For example, users can select the **OPEN** action on a file object to start an editor to work on the file and drag the file object and drop it on a printer object to print it.

The concept of applications is transparent to the user. Users interact with objects rather than with the operating system or separate applications. They focus on the task and not on the tools used to perform the task. Users interact with objects in the same manner across tasks.

New Features for OS/2 2.0

The OS/2 Workplace Shell provides an object-oriented user environment based on the 1991 CUA guidelines. In the OS/2 2.0 Workplace Shell, applications from OS/2 1.x are replaced by objects and collections of objects or folders. Objects correspond to devices (such as printers), applications (.EXEs), and file objects (such as files, subdirectories, and drives). Objects can be organized by users into folders, which are also objects.

Object names or labels can be equivalent to the objects they represent. For example, the object that represents the program file MYAPP.EXE can be named MYAPP.EXE. Object names can also be changed to whatever is meaningful. For example, the object that represents the program file MYAPP.EXE can be relabelled "My Application." Where an object resides is also transparent to users. For example, an object can represent a printer device that is physically attached to the user's system or attached through a network. Figure 1 shows the desktop in the new object-oriented user environment.

Some of the objects the Workplace Shell provides are described in Table 1. After installation of OS/2 2.0, some of them appear directly on the Desktop. Some of them are contained in folders. Users can rearrange and relabel them to suit themselves.

WORKPLACE CLASSES

Every user object in the Workplace Shell is an instance of a Workplace software class object. There is a one-to-one correspondence between Workplace (user) objects and Workplace (software) classes. This is evident in the class hierarchy for Workplace objects, shown in Figure 2.

Workplace classes are defined using SOM's Object Interface Definition Language (OIDL). Workplace class libraries are built using the SOM compiler. Workplace objects are instantiated by the Workplace on behalf of the user through the Workplace Class List Object, installation programs, or batch files. The same rules that apply to SOM classes apply to Workplace classes, with the exception that applications cannot call Workplace methods. (SOM client applications can call SOM methods.)

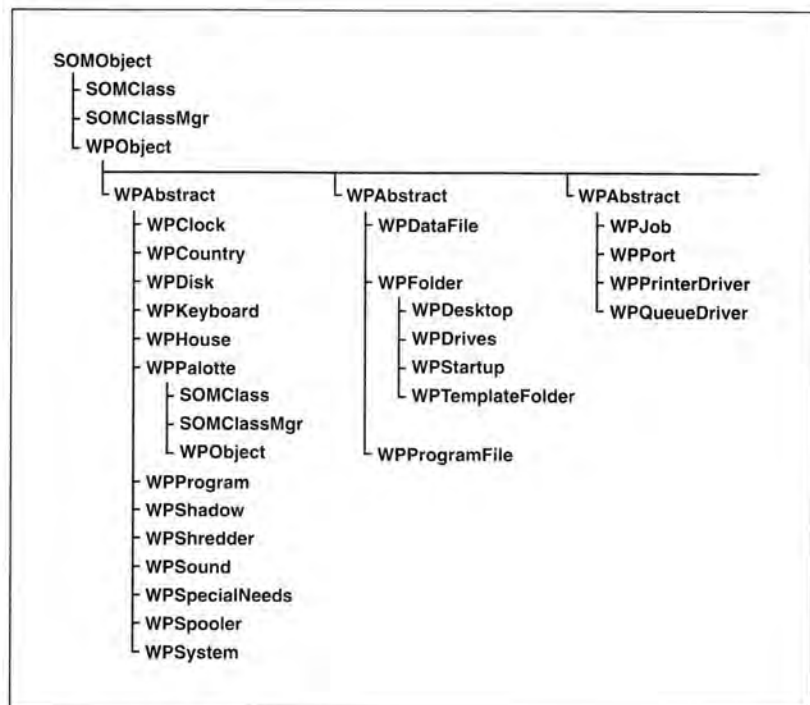


Figure 2: Hierarchy of Workplace objects

Characteristics and behavior for objects belonging to a class are defined as new methods for the object's class as well as methods inherited from ancestor classes. Workplace objects can take advantage of characteristics and behavior provided by existing Workplace classes by subclassing



SOME WORKPLACE OBJECTS PROVIDED BY OS/2 2.0

Object	Purpose
Clock	Set and view the current date, time, and alarm
Country	Set and view international conventions for system elements (country, date,time, and numbers)
Keyboard	Set and view keyboard configuration (timing, mappings, and special needs)
Mouse	Set and view behavior of mouse device (timing, setup, and button mappings)
Scheme Palette	Set and view window color and font attributes
Font Palette	Set and view fonts for textual elements of user interface and apply fonts to windows
Color Palette	Set and view colors for visual elements of user interface and apply color to a window
Printer	Set and view a print destination (a print queue and its associated port)
Shredder	Destroy an object
Sound	Enable or disable warning beep
Special Needs	Explain implications of special needs mode when activated
Spooler	Enable or disable spooling; set and view spool path
System	Set and view behavior of system elements (confirmations, logo, and windows)

Table 1: Workplace objects provided by OS/2 2.0

WORKPLACE STORAGE CLASSES

Class	Storage of Object Instance Data
WPAbstract	Object instance data stored in user profile (OS2.INI)
WPFileSystem	Object instance data stored in files in the file system
WPTransient	Object instance data not saved

Table 2: Workplace storage classes



those classes or deriving their new class from the existing classes.

All Workplace classes are derived from a root Workplace class, **WPObj**ect, which is derived from the root SOM class, **SOMObj**ect. **WPObj**ect defines behavior common to all Workplace objects. The immediate descendants of **WPObj**ect are storage classes from which all other Workplace classes are derived. Storage classes define methods for storing and retrieving data associated with instances of descendant classes. Storage classes provided with the Workplace Shell are shown in Table 2.

Objects whose instance data and state are preserved between system shutdown and system startup are called persistent objects. Objects whose instance data and state need not be preserved between system shutdown and system startup are called nonpersistent objects.

DESIGNING WORKPLACE CLASSES

To design a Workplace class, first identify all the actions to which an object instance can respond. Based on this list, define the methods in the class definition file that correspond to the actions identified. To illustrate this process and understand how method requirements for a class are identified, consider the following general description of objects and how it is used to define the **WPObj**ect class.

Based on the general description of user objects in a CUA-conforming user environment, objects have:

- Properties (for example, an icon representation on the Desktop)
- Views that contain information about the object
- Context or pop-up menus that describe actions to which the object can respond
- Mobility (objects can be directly manipulated).

Because **WPObj**ect defines general characteristics and behaviors common to all Workplace objects, these characteristics and behaviors should be reflected in the methods in the class

definition file for the **WPObj**ect class. Table 3 shows the mapping of characteristics and behaviors common to all Workplace objects to method groups defined for the **WPObj**ect class.

SETTINGS NOTEBOOK METHODS

The **WPObj**ect class provides for a standard page in a Workplace object's settings notebook called a **General** page. This page describes all the general object properties (title, icon, and a user-definable setting to specify whether or not this object is a template). All Workplace objects have general properties associated with them; therefore, all Workplace objects have a **General** page in their settings notebook.

DEFINING METHODS FOR THE WP OBJECT CLASS

Method Group	Characteristic or Behavior
Settings Notebook	Describe properties of an object
Save and Restore State	Persistence of Object Instance Data
Object Usage	Object Usage Information
Pop-up Menu	Actions that users can perform on objects
Set and Query	Object information (views, style, and title)
Error Handling	Error returns from object methods
Memory Management	Memory allocation for object resources
Setup and Cleanup	Object initialization and termination
Direct Manipulation	Object mobility (drag and drop)

Table 3: Defining methods for the Workplace object class

Settings notebook pages for Workplace objects are inherited from ancestors of their Workplace class. They include pages that have been added or removed by ancestor classes, in

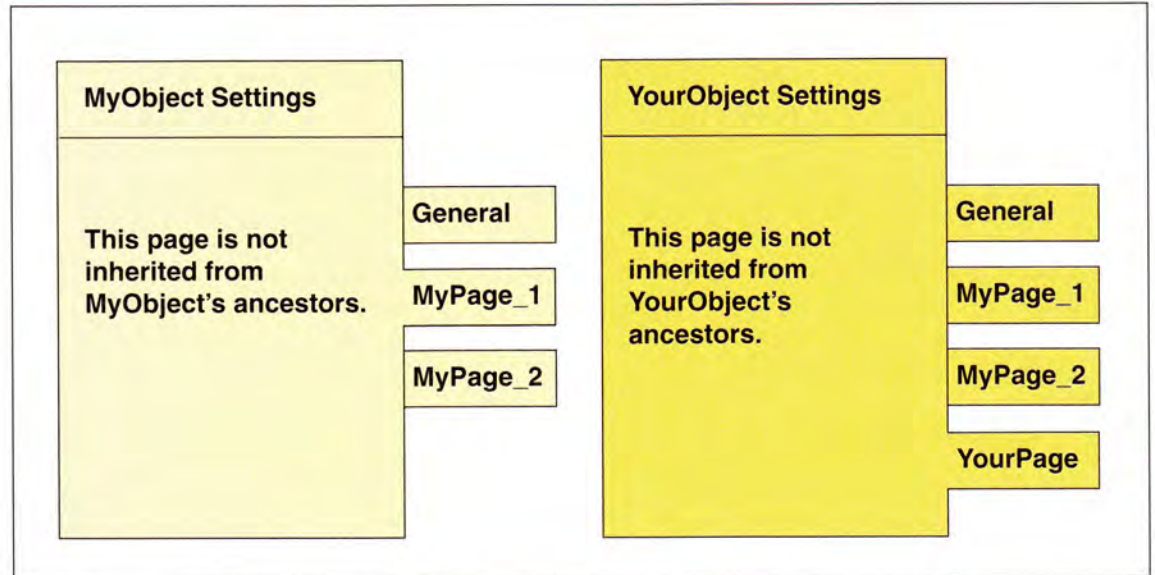


Figure 3: Settings notebook pages for **MyObject** and **YourObject**

SETTINGS NOTEBOOK METHODS

Method	Description
<code>wpAddObjectGeneralPage</code>	Add the General page to the object's settings notebook
<code>wpAddSettingsPages</code>	Add pages to the object's settings notebook
<code>wpInsertSettingsPage</code>	Insert a page into the object's settings notebook

Table 4: Settings notebook methods

addition to the **General** page inherited from the root **WPObject** class. For example, suppose that **MyObject** is a class of persistent objects derived from the **WPAbstract** class. Because **WPAbstract** is derived from **WPObject**, **MyObject** inherits characteristics and behaviors from **WPAbstract** and **WPObject**. **WPAbstract** inherits its settings notebook from **WPObject**. **MyObject**, therefore, also inherits **WPObject**'s settings notebook, which consists of a general page.

Suppose **MyObject** defines two additional pages in its settings notebook: **MyPage_1** and **MyPage_2**. This means that the settings notebooks for **MyObject** have three pages: **General**, **MyPage_1**, and **MyPage_2**. Now suppose **YourObject** is derived

from **MyObject** and, therefore, by inheritance, instances of **YourObject** have a **General** page as well as **MyPage_1** and **MyPage_2** in their settings notebooks. Also suppose **YourObject** defines an additional page in its settings notebook: **YourPage**. The settings notebook for **YourObject** has four pages: **General** (inherited from the **WPObject** class), **MyPage_1**, **MyPage_2** (inherited from **MyObject**), and **YourPage**. The pages in the settings notebooks for **MyObject** and **YourObject** are shown in Figure 3.

The settings-notebook methods, as shown in Table 4, allow you to add new pages or remove pages a class has inherited from its ancestor's settings notebook.



When a user requests the object's settings view, the shell builds the object's settings notebook by calling the object's `wpAddSettingsPage` method. If the object shares the settings notebook of its parent class, the `wpAddSettingsPages` method belonging to the object's parent class is called.

New pages are added to the settings notebook an object inherits from its ancestors by overriding the object's `_wpAddSettingsPage` method and:

- Calling a new method that inserts the new page.
- Calling the `wpAddSettingsPage` method belonging to the object's parent class (`parent_wpAddSettingsPage`).

The new method calls the `wpInsertSettingsPage` method to insert the new page into the object's notebook. This technique is shown in Figure 4.

New pages for an object's settings notebook can be placed at the top or bottom of pages inherited from the object's ancestors. By calling the `parent_wpAddSettingsPage` method before calling the new method (`wpAddAnotherPage`), the new page is added to the top of the settings notebook, before pages inherited from the object's ancestors. If the sequence is reversed, the new page is added to the bottom of the settings notebook, after the pages inherited from the object's ancestors.

A page can be removed from an object's settings notebook by overriding the ancestor's method that inserts it. From the previous example, the settings notebook for `YourObject` inherits the pages (`General`, `MyPage_1`, and `MyPage_2`) defined by its ancestors (`MyObject` and `WPObject`). But `YourObject` may have requirements for `MyPage_1` but not `MyPage_2`. To remove `MyPage_2` from `YourObject`'s settings notebook, `YourObject` must override the method inherited from `MyObject` that adds `MyPage_2` to the settings notebook and return `SETTINGS_PAGE_REMOVED` without calling the parent method. This method is illustrated in Figure 5.

The same technique can be used to replace or remove the `General` page from an object's settings notebook by overriding the `wpAddSettingsPages` and `wpAddObjectGeneralPage` methods. The override to `wpAddSettingsPages`

calls the `wpAddObjectGeneralPage` method. To remove the `General` page, the override to `wpAddObjectGeneralPage` returns `SETTINGS_PAGE_REMOVED` without calling the parent method. To replace the `General` page with another page, the override to `wpAddObjectGeneralPage` calls `wpInsertSettingsPage` without calling the parent method.

```

/***** New Methods *****/
SOM_Scope ULONG SOMLINK MyObject_wpAddAnotherPage(MyObject *somSelf,
                                                    HWND hwndNotebook)
{
    PAGEINFO pageinfo;
    .
    /* Set up page information data structure for my new page. */
    .
    /* Add the page to the Settings Notebook */
    return _wpInsertSettingsPage(somSelf, hwndNotebook, &pageinfo);
}

/***** Method Overrides *****/

SOM_Scope BOOL32 SOMLINK MyObject_wpAddSettingsPage (MyObject
*somSelf,
                                                    HWND hwndNotebook)
{
    .
    if (parent_wpAddSettingsPage (somSelf, hwndNotebook)
        && _wpAddAnotherPage (somSelf, hwndNotebook))
        return (TRUE);      /*pages added successfully */
    else
        return (FALSE);     /*something failed -return error */
}

```

Figure 4: Adding pages to an object's settings notebook

```

/***** Method Overrides *****/
SOM_Scope ULONG SOMLINK YourObject_wpAddAnotherPage(YourObject
*somSelf,
                                                    HWND hwndNotebook)
{
    .
    /* Remove this page from the Settings notebook. */
    return (SETTINGS_PAGE_REMOVED);
}

```

Figure 5: Removing a page from an object's settings notebook



POP-UP MENUS

Pop-up menu methods support the actions users can perform on an object. These actions appear in a context or pop-up menu when the user presses button 2 of the pointing device. A pop-up menu, as shown in Figure 6, contains action choices for an object in its current

Open (->)	x Settings
Help	Details
Print (->)	

Figure 6: Pop-up menu options

POP-UP MENU METHODS TO MODIFY AN OBJECT'S POP-UP MENU

Method	Description
<code>wpFilterPopupMenu</code>	Filter out options from object's pop-up menu that don't apply
<code>wpInsertPopupMenuItems</code>	Insert items into object's pop-up menu item
<code>wpModifyPopupMenu</code>	Add new options to the object's pop-up menu

Table 5: Pop-up menu methods to modify an object's pop-up menu

```

/*                      Method Override                      */

/*  Filter out any options from the pop-up that don't apply. */

SOM_Scope ULONG  SOMLINK MyObject_wpFilterPopupMenu(MyObject *somSelf,
                                                    ULONG ulFlags, HWND hwndCnr, BOOL32 fMultiSelect)
{
    MyObjectData *somThis = MyObjectGetData(somSelf);
    MyObjectMethodDebug("MyObject", "MyObject_wpFilterPopupMenu");

    /* Don't allow anyone to print MyObject                      */

    return( parent_wpFilterPopupMenu(somSelf, ulFlags, hwndCnr, fMultiSelect)
           & ~CTXT_PRINT );
}

```

Figure 7: Removing standard items from an object's pop-up menu

context, or state. The contents of a pop-up menu depends on the state of the object.

Pop-up menus consist of a set of selectable items and any pull-down or conditional cascade menus associated with them. In Figure 6, **Open**, **Help**, and **Print** are items in the object's primary pop-up menu. **Settings** and **Details** are items in the **Open** pull-down or conditional cascade menu. Conditional cascade menus have mini buttons displayed next to the pop-up menu item. If users select the pop-up menu item, a default action listed in the pull-down menu is performed. If users select the mini button, the pull-down menu is displayed. It is a conditional cascade menu because it is displayed only if the user selects the mini button.

Like settings-notebook pages, pop-up menus are inherited from a class's ancestor classes. They include pop-up menu items ancestor classes have added to or removed from the pop-up menu inherited from `WPObj`. The pop-up menu methods shown in Table 5 allow users to add or remove new menu items from the pop-up menu inherited from an object's ancestor classes.

When users request an object's pop-up menu, the Shell builds the object's pop-up menu by calling the object's `wpFilterPopupMenu` and `wpModifyPopupMenu` methods. The `wpInsertPopupMenuItems` method is called by an override to the `wpModifyPopupMenu` method to add new options to an object's pop-up menu.

Adding and Removing Items from a Pop-Up Menu

The `WPObj` class defines a set of standard pop-up menu items that are inherited by all Workplace objects. The pop-up menu of a Workplace object consists of a subset of the standard pop-up menu items and any new menu items defined for the object's class or inherited from other ancestors.

Workplace classes can add or delete standard pop-up menu items from their pop-up menu by overriding the `wpFilterPopupMenu` method. Each standard pop-up menu item is associated with a flag defined by the `WPObj` class. The flags are represented by constant names prefixed by `CTXT_` and are defined in the toolkit header files.

The `wpFilterPopupMenu` method returns the flags that represent the pop-up menu items for the object. In removing a standard pop-up menu item from the pop-up menu, the override to `wpFilterPopupMenu` masks the flag that corresponds to the item being removed from the flags that represent the standard pop-up menu items inherited from the object's parent. For example, suppose that printing `MyObject` has no meaning. To remove the Print option from `MyObject`'s pop-up menu, `wpFilterPopupMenu` is overridden as shown in Figure 7.

In this example, flags that represent the standard pop-up menu items of `MyObject`'s parent class are returned from the call to `parent_wpFilterPopupMenu`. To remove the Print option from `MyObject`'s pop-up menu, these flags are ANDed with the complement of `CTXT_PRINT`. Conversely, if the pop-up menu of `MyObject`'s parent class did not include the Print option, it can be added to `MyObject`'s pop-up menu by ORing these flags with `CTXT_PRINT`.

An object's pop-up menu is inherited from its ancestors. To ensure that calls to the `wpFilterPopupMenu` method belonging to the object's ancestors do not add the menu item after it is deleted or remove the menu item after it is added, the `parent_wpFilterPopupMenu` method is called first.

Adding Class-Specific Items to the Primary Pop-Up Menu

New items are added to the pop-up menu inherited from an object's ancestors by:

- Defining a new method that returns flags representing the new items being added to the object's pop-up menu.
- Overriding its `wpModifyPopupMenu` method and calling the `wpInsertPopupMenuItems` method if the flag associated with the menu item is set.

For example, to add "New Item" to `MyObject`'s pop-up menu, the new menu item is defined in a resource file in the same manner as menus are defined in PM programs. An ID is assigned to the new menu and to the menu item, as shown in Figure 8.

IDs for class-specific menus and menu items have a value greater than `WPMENUID_USER`, so they do not conflict with IDs for menus and menu

items defined by the Workplace classes provided with the Shell.

A new method for `MyObject`, `wpMyFilterPopupMenu`, returns a flag that represents the new item added to the inherited pop-up menu for `MyObject`. The override to `wpModifyPopupMenu` uses

```
#define ID_MOREITEMS    WPMENUID_USER+1

#define ID_NEWITEMS     WPMENUID_USER+2


MENU ID_MOREITEMS LOADONCALL MOVEABLE DISCARDABLE
BEGIN
    MENUITEM "New Item", ID_NEWITEMS
END
```

Figure 8: Resource file defining new items for object's pop-up menu

```
/* Define MYCTXT_NEWITEM here */

/*          New Method */
/* Filter new items added to MyObject's pop-up menu */
SOM_Scope ULONG SOMLINK MyObject_wpMyFilterPopupMenu(MyObject*somSelf,
    ULONG ulFlags, HWND hwndCnr, BOOL32 fMultiSelect)
{
    .
    return(MYCTXT_NEWITEM);
}

/*          Method Override */
/* Add a new item to MyObject's pop-up menu */

SOM_Scope BOOL32 SOMLINK MyObject_wpModifyPopupMenu(MyObject
*somSelf,
    HWND hwndMenu, HWND hwndCnr, ULONG ulPosition)
{
    .
    /* Get flags for new items for MyObject's pop-up menu */
    flags=_wpMyFilterPopupMenu(...);

    /* Insert new items in MyObject's Primary Menu if flag is set */
    if (flags & MYCTXT_NEWITEM)
        _wpInsertPopupMenuItemsA(somSelf, hwndMenu, ulPosition, hmod,
            ID_MOREITEMS, WPMENUID_PRIMARY);

    /* Add the items inherited from MyObject's parent */
    return (parent_wpModifyPopupMenu(somSelf,hwndMenu,hwndCnr,ulPosition));
}
```

Figure 9: Adding class-specific items to an object's pop-up menu



```
/* Insert new items in MyObject's Open SubMenu if flag is set */
if (flags & MYCTXT_NEWITEM)
    _wpInsertPopupMenuItemsA(somSelf, hwndMenu, ulPosition, hmod,
                            ID_MOREITEMS, WPMENUID_OPEN);
```

Figure 10: Adding an item to a pop-up menu submenu

```
/*                      Method Override                      */
/*  Filter new items added to MyObject's pop-up menu          */
SOM_Scope ULONG SOMLINK YourObject_wpMyFilterPopupMenu(
    YourObject *somSelf, ULONG ulFlags, HWND hwndCnr, BOOL32 fMultiSelect)
{

    return(0); /* Return no flags so that 'New Item' is not    */
              /* included in YourObject's pop-up menu          */
}
/*                      */
```

Figure 11: Removing class-specific items from a pop-up menu

POP-UP MENU METHODS THAT SUPPORT NEW POP-UP MENU ITEMS

Method	Description
<code>wpMenuItemHelpSelected</code>	Display the help associated with class-specific pop-up menu item
<code>wpMenuItemSelected</code>	Process class-specific pop-up menu items

Table 6: Pop-up menu methods that support new pop-up menu items

the flags returned from `wpMyFilterPopupMenu` to determine which items are inserted in `MyObject`'s pop-up menu. This technique is illustrated in Figure 9.

The `wpInsertPopupMenuItems` method requires a handle to the module where the menu resource is defined, the ID for the menu resource, and the ID for the menu where the item is being inserted. In this example, `ID_MOREITEMS` is the ID for the menu resource that defines the new menu item being added to the object's primary pop-up menu.

`WPMENUID_PRIMARY` is the ID for the object's primary pop-up menu, where "New Items" is being inserted.

An item can be added to a pop-up menu's submenu or conditional cascaded menu by specifying the ID for the conditional cascaded menu on the call to `wpInsertPopupMenuItems`. For example, to add "New Items" to the `Open` conditional cascaded menu, the call to `wpInsertPopupMenuItems` is modified as shown in Figure 10.

WORKPLACE API

Function	Description
<code>WinCreateObject</code>	Create an object
<code>WinDeregisterObjectClass</code>	Deregister (remove) a Workplace object class
<code>WinDestroyObject</code>	Delete a Workplace object
<code>WinEnumObjectClasses</code>	Get a list of all Workplace classes that have been registered
<code>WinQueryObject</code>	Get a handle to a given object
<code>WinRegisterObjectClass</code>	Register a Workplace object class
<code>WinReplaceObjectClass</code>	Replace a registered class with another registered class
<code>WinSetObjectData</code>	Change settings on an object that was created with <code>WinCreateObject</code>

Table 7: Workplace API

REXX UTILITY WORKPLACE FUNCTIONS

Function	Description
<code>SysRegisterObjectClass</code>	Register a Workplace object class
<code>SysDeregisterObjectClass</code>	Deregister a Workplace object class
<code>SysQueryClassList</code>	Get the list of Workplace class registered with the Shell
<code>SysCreateObject</code>	Create a Workplace object

Table 8: REXX utility workplace functions

Removing Class-Specific Items from a Pop-Up Menu

Class-specific pop-up menu items inherited from ancestor classes are removed by overriding the methods that return flags representing those items. For example, suppose `MyObject` is the parent class of `YourObject`, but `YourObject` has no requirement

for the "New Item" in the pop-up menu it inherits from `MyObject`.

To remove "New Item" from `YourObject`'s pop-up menu, `YourObject`'s class overrides `wpMyFilterPopupMenu` and does not return the `MYCTXT_NEWITEM` flag that `MyObject` defined for "New Item." This technique, shown in Figure 11, is similar to that described for removing



standard items from an object's pop-up menu. It requires that `MyObject` publish `wpMyFilterPopupMenu` as well as the `MYCTXT_NEWITEM` flag so that subclasses of `MyObject` can remove or add "New Item."

Supporting User Selection of New Pop-Up Menu Items

When a class defines new actions for its pop-up menu, it must provide for the processing of the actions when the user selects them. This is done by overriding the pop-up menu methods shown in Table 6.

SUPPORTING USER SELECTION OF NEW POP-UP MENU ITEMS

```
/*                      Method Override                      */
/* Process input from the extra menu option that we added. */

SOM_Scope void  SOMLINK MyObject_wpMenuItemSelected(MyObject *self,
                                                    HWND hwndFrame, ULONG MenuID)
{
    .
    /* Which of our menu items was selected ? */
    switch( MenuID )
    {
        case ID_SUBITEM1:
            .
        case ID_SUBITEM2:
            .
        case ID_SUBITEM3:
            .
        case ID_SUBITEM4:
            .
        default:
            parent_wpMenuItemSelected
    }
}
```

Figure 12: Supporting user selection of new pop-up menu items

Using the previous example, `MyObject` supports `SubItem_1` by overriding the `wpMenuItemSelected` method as shown in Figure 12.

`MyObject` support helps for new pop-up menu items by overriding `wpMenuItemHelpSelected` in a similar manner.

THE WORKPLACE APPLICATION INTERFACE

Outside the Workplace environment, objects are dynamic link libraries that consist of data and code operating on that data when objects are instantiated in the Workplace (run-time) environment. Workplace objects have no life outside the Workplace environment.

Workplace classes come to life when the class is registered with the Workplace Shell and the class is instantiated. The Workplace Shell and SOM provide the underlying code (predefined Workplace object methods) that supports an object's existence. The Shell calls the appropriate object methods when the user interacts with the object. In this sense, the Shell is the client of all Workplace objects. The Shell manipulates the object (its code) on behalf of its users.

Applications, on the other hand, cannot call Workplace object methods directly. They are not clients of Workplace objects in the same sense that applications can be clients of SOM objects. Workplace objects are derived from the `WPObject` class, which is derived from the `SOMObject` class. They share all the features of SOM objects: inheritance, polymorphism, and so on. But only the Shell can directly manipulate them.

Because there are times when applications may need to make changes to the Desktop and objects on the Desktop, the Workplace Shell provides functions that enable your application to effect those changes. These functions are summarized in Table 7.

REXX UTILITY WORKPLACE FUNCTIONS

The REXX programming language also provides utility functions for Workplace classes and objects. These functions are listed in Table 8.

REXX utility functions allow users to write batch files to operate on Workplace classes and objects in the same way as applications.

Installing a Workplace Object

Workplace objects can be installed on the Desktop in one of two ways:

- By running an installation program or batch file
- By using the Workplace class list object.

Application developers can provide installation programs for their objects. From the user's perspective, installation consists of putting a diskette in the diskette drive, double-clicking on the diskette drive, and then on the installation program.

An installation program is responsible for:

- Copying the dynamic link library (DLL) that contains the object's class definition from a disk to the \OS2\DLL directory or to a directory in \LIBPATH
- Registering the class and its DLL name with the Shell by calling `WinRegisterObjectClass`
- Creating an object instance of the class and placing it on the Desktop or in a particular folder by calling `WinCreateObject`.

Instantiating an object is an optional responsibility for an installation program. When a class is registered by calling `WinRegisterObjectClass`, an object template is placed in the Templates folder on the Desktop if the class supports templating. Users can create instances of these objects by tearing off a copy of the template. This strategy can be useful for larger applications that define data-file objects associated with program objects.

An installation batch file written in REXX performs the same operations, but uses the REXX utility functions `SysRegisterObjectClass`, `SysCreateObject`, and `SysDeRegisterObjectClass` instead of the `WinRegisterObjectClass`, `WinCreateObject`, and `WinDeRegisterObjectClass` Workplace functions.

The OS/2 2.0 toolkit provides a Workplace class list object that is automatically installed during installation of the toolkit. This object is a tool that provides a windowed user interface for general Workplace class registration and object creation activities. It performs all the functions a typical Workplace object

installation program must provide, with the exception of copying files from an installation diskette to a hard disk. It is a fast and easy way to build and test objects in an application-development environment. It is not a tool for the general OS/2 user who knows nothing about Workplace classes.

Mary Wright, IBM Personal Systems LOB, 1000 N.W. 51st St., Boca Raton, Fla., 33429. Mrs. Wright is a staff information developer in Boca Raton. She came to IBM in 1982 as a developer of the Customer Support System, having previously worked as an applications programmer specializing in engineering applications. In 1985, she joined OS/2 development and was the original developer of the Monitor Dispatcher. In 1988, she joined information development, where she is the technical editor for the OS/2 Programming Library. She was the lead writer for the Application Design Guide and the three-volume Programming Guide in the OS/2 2.0 Technical Library. Mrs. Wright received a B.A. in Mathematics and Russian and an M.A. in Russian from Case Western Reserve University.

REFERENCES

Chapter 8: "Workplace Programming Interface," *Application Design Guide*, OS/2 2.0 Technical Library.

Chapter 7: "Object-Oriented Programming Using SOM," *Application Design Guide*, OS/2 2.0 Technical Library.

System Object Model Guide and Reference, OS/2 2.0 Technical Library.

Presentation Manager Programming Reference Volume II, OS/2 2.0 Technical Library.

Sessions, Roger and Nurcan Coskun. "Object-Oriented Programming in OS/2 2.0," *IBM Personal Systems Developer* (Winter 1992): 107-119.

Sessions, Roger and Nurcan Coskun. "Class Objects in SOM," *IBM Personal Systems Developer* (Summer 1992) 67-77.

The OS/2 Toolkit also provides a file that contains code examples for some of the methods for the predefined Workplace classes. See the Toolkit Readme file for more information.



Multimedia and Graphics

DVI Support by Audio Visual Connection



Kai-ching Chu

by Kai-ching Chu

The ActionMedia II™ display board, based on the technology of Digital Video Interactive (DVI), can play back video and image files on the IBM PS/2 "on the fly." With the addition of the recently released Audio Visual Connection™ 1.05 software, users can combine digitally generated video and still images with VGA and XGA images, text, and APCA sound to create a complete presentation.

This article discusses the design of AVC-DVI support and of AVA, an extended authoring language that handles DVI file playback. It includes examples of how to control playback synchronization and achieve various dynamic audio and visual effects.

DVI, ACTIONMEDIA, AND AVC

IBM and Intel's recent advances in DVI technology enable users to capture, digitize, and compress audio and video signals. Once compressed, full-motion video signals can be played back from storage devices such as CD-ROM, rewritable optical disks, and magnetic hard disks.

With these advances and the 1992 introduction of IBM's PS/2 ActionMedia II adapter (which uses DVI processing chips for display, compression, and decompression), programmers can develop powerful and cost-effective digital video applications that convert the PS/2 into a multimedia desktop computer.

Audio Visual Connection™ (AVC) is an award-winning PS/2 program that lets users author and present multimedia demonstrations and applications. Through the M-Motion Video Adapter™, computer

graphics images and text can be combined with audio and analog video sources in AVC presentations. Beginning with AVC 1.05, support of DVI sources is provided through the ActionMedia II adapter board. With digitized video, audio or still image files, the new AVC program now provides a powerful and complete solution for your presentation needs. Digital video files are portable, can be copied and transmitted, and can be accessed from PCs, detachable storage devices, or LAN servers. AVC brings vivid and dynamic video to advertising, training, education, and other applications.

At the 1991 Comdex/Fall Show in Las Vegas, Nev., ActionMedia II hardware and software won "Best Multimedia Product" and "Best of Show" awards. AVC with DVI support has been instrumental in demonstrating DVI technology and products, as well as the potential for multimedia applications.

DIGITAL VIDEO AND AUDIO IN PRESENTATIONS

Without compression, a digitized video data stream would take up several hundred kilobytes to several megabytes per image frame, depending on the resolution and color depth of the images. For full motion video, there are up to 30 image frames per second in the data stream to be processed.

ActionMedia II handles two types of compressed video files, Production Level Video (PLV) and Real Time Video (RTV). Both have an average compression ratio of more than 100:1. While PLV files can be produced off line by Intel's digital compression facility, the lower quality RTV files can be captured

Programmers can develop powerful and cost-effective digital video applications that convert the PS/2 into a multimedia desktop computer.

CYBERARTS™

INTERNATIONAL

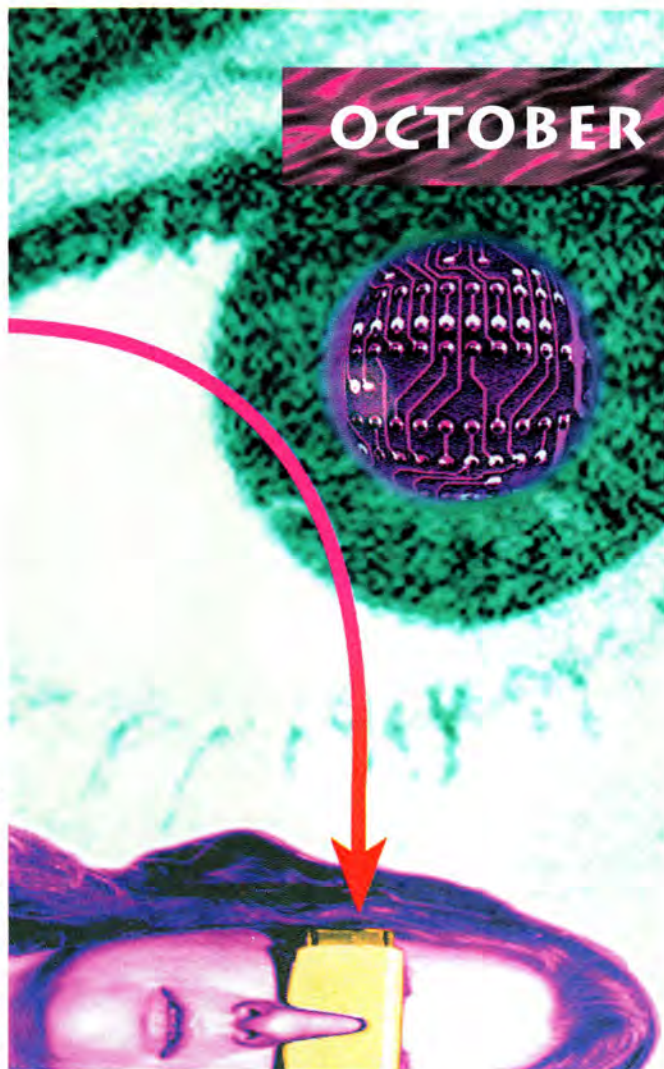
1992

OCTOBER 29 - NOVEMBER 1

PASADENA CENTER

CYBERARTS is the leading how-to conference featuring Master Classes in the technology and art of building Virtual Realities by the world's foremost experts on Computer Animation, Interactivity, User-Interface Design, Media Integration, Video, MIDI, 3D Sound and Graphics, Video and Audio Synthesis Techniques, CD-ROM Authoring, Digital Signal Processing, and Entertainment, Education, Film, Theme Park, and even Fine Art Applications, and much more. Plan to attend this cutting-edge forum and exhibition of interactive and multimedia technologies now!

ILLUSTRATION DESIGN: RICHARD LEEDS



The New Sciences & The New Arts Dance

FOR MORE INFORMATION ON ATTENDING OR EXHIBITING AT THE
THIRD ANNUAL CYBERARTS, WRITE CYBERARTS INTERNATIONAL, 20085
STEVENS CREEK BLVD., CUPERTINO, CA 95014. OR FAX 408-446-1088.



and compressed in real time by an IBM PS/2 with ActionMedia II. PLV and RTV files can be decompressed and played back "on the fly" on a PS/2.

DVI files' audio streams use Pulse Code Modulation (PCM, 8-bit samples) or Adaptive Differential Pulse Code Modulation (ADPCM, 4-bit samples) compression algorithms.

ActionMedia II still images can be stored compressed or uncompressed, in 9- and 16-bit per pixel formats. ActionMedia II also supports a 9-bit baseline format of the Joint Photographers Expert Group (JPEG) Standard, which is adopted by the International Standards Organization.

HOW DVI AND AVC WORK TOGETHER

AVC program support for DVI was built with the Audio Video Kernel (AVK), an application programming interface for ActionMedia II. OS/2's multithreaded programming technique and dynamic link libraries are used in AVC to control different audio and video sources, which can be used in synchronous multimedia presentations.

A complicated presentation might combine VGA and XGA images with DVI still images and DVI video (which require decompression on the fly). Audio can come from either the Audio Capture and Playback Adapter (ACPA) or DVI. All of these sources need to be well coordinated and synchronized.

When DVI images are displayed on a PS/2 monitor already showing a VGA or XGA image, the DVI images appear on a plane behind the VGA/XGA image, "showing through" any transparent areas of that image. These two planes allow overlay of image, text or video. Multithreaded programming techniques allow efficient switching between several computing tasks. While playing the video or audio for AVC, for example, a user can manipulate overlay text or images on the screen or alter certain video and audio control parameters. Figure 1 shows how text and images can be layered on the screen.

DVI pictures can be either displayed on the PS/2 screen or directed to a TV monitor through the S-video output line of the ActionMedia II Adapter. In the latter mode, AVC story lines and VGA/XGA images are shown on the PS/2 monitor while DVI video or still images are viewed on a separate TV or transmitted to a VCR.

Multithreaded programming techniques allow efficient switching between several computing tasks.



Figure 1: AVC's dual display for DVI video/image overlay

AVA: THE AUTHORING LANGUAGE OF AVC

Using AVC's Story Editor, AVC stories are written in the Audio Visual Authoring (AVA) language, a high-level interpretive language for assembling lines of logic describing audio-visual presentations. The operators and instructions of AVA are a subset of REXX. The AVC package has a run-time module to run the stories.

Since the AVA language is interpretive, AVC story writers can create and modify their presentations interactively. AVA has logical instructions and structured programming capabilities to display and control media and direct the flow of stories. Internal or external macro subroutines can be used repetitively or within controlled phrases.

Prior to the release of AVC 1.05, the basic AVA instructions were **PLAY** for ACPA audio files and **SHOW** for images (MCGA, VGA, and XGA). With the addition of other AVA instructions, AVC has the power to construct complex stories and link to external environments. It can also call external programs, mainframe connections, and SQL database services.

DVI INSTRUCTIONS INCLUDED WITH AVA

With DVI support added to AVC 1.05, the AVA language now includes:

- **DPLAY** plays a DVI file (video and audio).
- **DSHOW** displays a DVI still image.
- **DCONFIG** controls the DVI playing configuration.
- **DLOCATE** defines source and target image size and location.
- **DPAUSE** pauses a video sequence.
- **DQUERY** returns the current video frame number and play status.
- **DSTATE** saves current or restores previous state values.
- **DSTEP** advances the DVI video one frame.
- **DSTOP** stops the DVI video or audio.
- **WAIT D** waits for a specific video frame number.

Figure 2 shows a simple use of **DPLAY**. The image is launched until a key is pressed.

```
DPLAY "C:\VIDEO\AIRPLANE.AVS"
WAIT KEY
```

Figure 2: Simple DVI video play

Figure 3 shows the use of **DPLAY** after an AVC image is displayed with the **SHOW** command. The AVC image exists as an overlay on top of the video image; the story continues after a key is pressed.

```
SHOW "C:\IMAGE\CLOUD" {WAIT= 0}
DPLAY "C:\VIDEO\AIRPLANE.AVS"
WAIT KEY
```

Figure 3: Simple DVI video play, overlay with image

A partial AVC image or text can also be overlaid with DVI video by using the **PASTE** command.

DSHOW for DVI still images has syntax similar to that of **DPLAY**. Specify the name of a DVI still file following the **DSHOW**. Any of the acceptable still file types can be used.

DCONFIG controls the DVI playing configuration, including:

- **Resolution** sets screen resolution to either 512x480 or 256x240.
- **Speed** controls video play speed.
- **Monitor** controls monitor use. DVI and VGA/XGA image can be merged on one monitor or viewed on two separate monitors.
- **Volume** controls audio volume.
- **Balance** controls audio balance between left and right channels.
- **A_V** selects presentation of video, audio, or both.



Since the AVA language is interpretive, AVC story writers can create and modify their presentations interactively.



- **Erase** determines whether a previously displayed DVI file image is erased from the screen when new images appear.
- **Contrast** adjusts the monitor contrast for DVI images.
- **Saturation** adjusts monitor saturation for DVI images.
- **Tint** adjusts monitor tint for DVI images.

```
DCONFIG CON (-100) /* Initial contrast at -100
DCONFIG SAT (-100) /* Initial saturation at -100
DPLAY "C:\VIDEO.AVS" /* Start video
Do i = -100 to 0 /* Fade-in video
    DCONFIG CON i /* change of contrast
    DCONFIG SAT i /* change of saturation
    WAIT .1
END
```

Figure 4: Fade-in video

```
DPLAY "C:\AIRPLANE.AVS"
Do until @DSTATUS=STOP /* Query for status during the
    DQUERY /* the entire play
    say "Frame no. = "@dFrame /* Display known frame number
    WAIT .1
End
```

Figure 5: Use of DQUERY

```
DLOCATE S 0 0 100 100 /* Default full size source
DLOCATE S 25 25 50 50 /* Middle quarter of the source
DLOCATE T 0 0 100 100 /* Default full screen target
DLOCATE T 25 25 50 50 /* Display in the middle quarter
/* of the screen

DO i=0 to 50
    DLOCATE T i i 100-i-i 100-i-i /* Shrink the target video
    WAIT .1 /* from full screen
END /* to a single dot

DO i = 0 to 50
    DLOCATE T i ((i-50)*(i-50))/25 i i /* curved path of
    WAIT .1 /* target rectangle
END
```

Figure 6: Various use of DLOCATE for display areas

- **Brightness** adjusts monitor brightness for DVI images.

Figure 4 shows fade-in during a DPLAY, using the manipulation of contrast and saturation in a DO-LOOP.

DQUERY returns the current frame number and play status in the AVA variables; @DFRAME: returns the current frame number and @DSTATUS: shows player status—STOP, PAUSE, PLAY, or ERROR.

In Figure 5, DQUERY is used to display frame numbers during video playback.

DLOCATE defines which part of a DVI source image to display as well as the size and location of the display area for the target image. Stills and video clips can be controlled by DLOCATE. DLOCATE S controls the image or video source; DLOCATE T controls the display target area.

The screen is divided into coordinate space of 100x100. The four parameters (x, y, dx, and dy) following DLOCATE S or DLOCATE T refer to the upper left coordinates, width, and height of the display area.

Combined application of DLOCATE S and DLOCATE T can create many interesting display schemes for video and still images. Figures 6 and 7 give simple examples.

SAMPLE MACROS FOR VARIOUS PRESENTATION EFFECTS

AVA's macro routine calls can be used to write interesting video and still image display, transition, or dissolve methods. This section presents additional examples for beginning AVC users.

In Figure 8, DVI still or video images are displayed in an array of small rectangles, each one-tenth the width and height of the screen.

Figure 9 shows the display transition routines SweepH, FlipH, SweepV, and FlipV.

A still image Still1 is shown in the middle of the screen. Still2 is then superimposed with the horizontal sweeping-across routine, followed by image Still3 with the horizontal page-

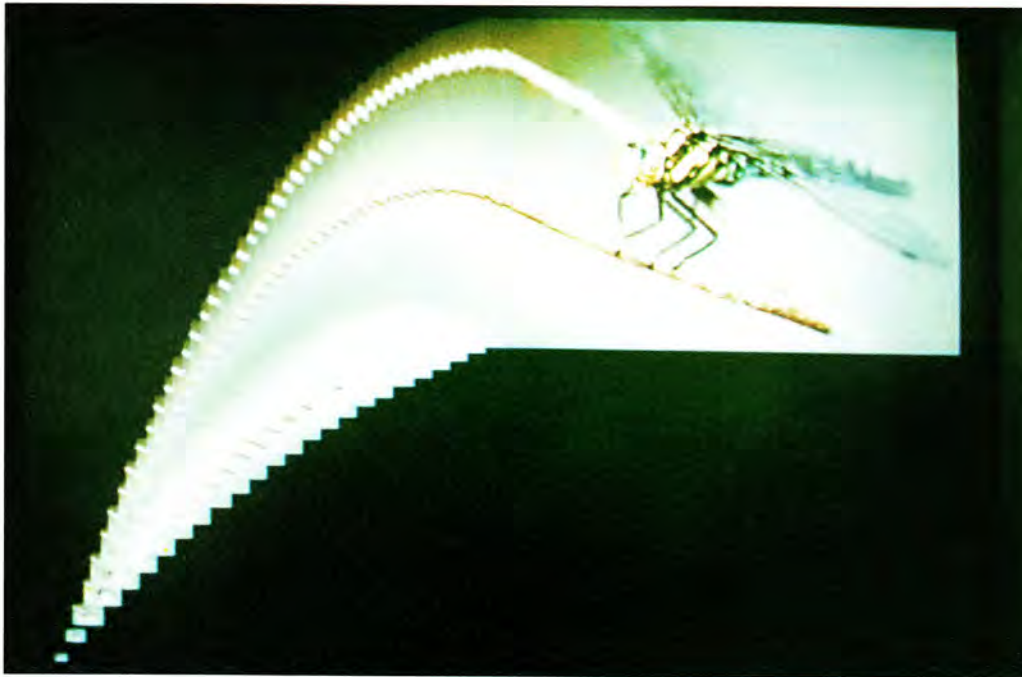


Figure 7: Last example in Figure 6, curved display path

flipping routine. Users can construct other methods such as vertical sweeping and flipping, band-sweeping, and tile-filling. These display transition methods can also be applied to video using **DPLAY**.

In general, the video **DPLAY** instruction is asynchronous and interruptible. While the video is run, the story is controlled by instructions such as **SHOW**, **PASTE**, **PLAY**, **DLOCATE**, **DCONFIG**, and **DQUERY**. The computer switches automatically between playing the video and running the other instructions. If the other instructions monopolize computer time, they could affect video synchronization. The video will be out of sync if given an insufficient time share. Instructions such as **WAIT** can be used to pace story flow and video run time. There is also a synchronous option to launch the video clips in the stories.

NEW METHODS OF COMMUNICATION

AVC support for DVI is built with the Audio Video Kernel (AVK), which is designed to work with future digital video hardware releases. AVK is being updated for OS/2 2.0, Microsoft Windows, and higher-level

multimedia operating system extensions. AVC written with AVK is in a stable environment and the porting path to other operating systems' extensions is relatively easy.

AVC addresses the main requirements of multimedia applications. With digital video storage and networking, applications built with AVC can be effectively customized, updated, and revised, leading to the flourishing of multimedia applications for business, travel, merchandising, manufacturing, entertainment, and education.

```
CALL GRID10 DSHOW "D:STILL.AVS"
CALL GRID10 DPLAY "D:VIDEO.AVS"
```

Grid10:

```
DLOCATE T 0 0 10 10
ARG(1) ARG(2)
DO j = 0 to 9
  DO i = 0 to 9
    DLOCATE T i*10 j*10 10 10
  END
END
RETURN
```

Figure 8: GRID10: Multiple display in an array of 10x10



```

DLOCATE T 25 25 50 50
DSHOW Still1          /* Display Still1 in the middle
wait 1                /* quarter of the screen
CALL SweepH DSHOW Still2 /* Replacing by Still2 with
wait 1                /* horizontal sweep
CALL FlipH DSHOW Still3 /* Replacing by Still3 with
wait 1                /* horizontal flip

```

```

SweepH:                /* Horizontal Sweep

```

```

DLOCATE T 25 25 0 50
DLOCATE S 0 0 0 100
ARG(1) ARG(2)
DO i = 0 to 50 by 5
  DLOCATE T 25 25 i 50
  DLOCATE S 0 0 i+i 100
  WAIT .1

```

```

END
RETURN

```

```

FlipH:                 /* Horizontal Flip

```

```

DLOCATE S 0 0 100 100
DLOCATE T 25 25 0 50
ARG(1) ARG(2)
DO i = 0 to 50
  DLOCATE T 25 25 i 50
  WAIT .1

```

```

END
RETURN

```

```

SweepV:                /* Vertical Sweep

```

```

DLOCATE T 25 25 50 0
DLOCATE S 0 0 100 0
ARG(1) ARG(2)
DO i = 0 to 50 by 5
  DLOCATE T 25 25 50 i
  DLOCATE S 0 0 100 i+i
  WAIT .1

```

```

END
RETURN

```

```

FlipV:                 /* Vertical Flip

```

```

DLOCATE S 0 0 100 100
DLOCATE T 25 25 50 0
ARG(1) ARG(2)
DO i = 0 to 50
  DLOCATE T 25 25 50 i
  WAIT .1

```

```

END
RETURN

```

Figure 9: SweepH, FlipH, SweepV, FlipV: display transition examples

ACKNOWLEDGMENTS

The author wishes to thank Barbara Plum and Tom Almeida for reviewing this article.

Kai-ching Chu, IBM Entry Systems Technology, 1055 Joaquin Rd., Mountain View, Calif. 94043, (415) 694-3023. Dr. Chu is a senior programmer currently responsible for digital video application development. He joined IBM in 1973 and has held various research, development, and managerial positions in Research, Programming Systems, and Personal Systems. He holds a Ph.D. in applied mathematics from Harvard University.

REFERENCES

Audio Visual Authoring Language Reference
(IBM Doc. 90X8367)

ActionMedia II Technical Reference
(IBM Doc. 92F2729)

File README.DVI in AVC 1.05



DON'T BELIEVE the HYPE!

FIND OUT ABOUT THIS
CUTTING-EDGE COMPUTING
TECHNOLOGY WITH



VR Special Report



From the publishers of the premier publication of the AI marketplace, this Special Report is your comprehensive directory to the cutting-edge of computer technology: **VIRTUAL REALITY.**

We'll show you the toys AND the tools, where to study virtual reality, what's available now, and what's coming soon. All this for only \$9.95!

To order your Virtual Reality Special Report, please call (415) 905-2332 or write to:

AI EXPERT
VIRTUAL REALITY SPECIAL REPORT
P.O. Box 2156
Knoxville, IA 50197 - 2156

Group Dynamics



Award-winning workgroup dynamics, from N/Joy. The first multifunctional, object-oriented business software designed for OS/2. Supporting development of complex documents on your LAN. With easy sharing of all text, spreadsheet and graphic data — all updated automatically via point-and-click hot links. Call 1-800-392-7724 now for a limited-time offer too dynamic to miss.



THE WORLD OF OBJECTS®

Vienna Software Publishing, 1398 SW 15th Street, Boca Raton, FL 33486, 1-800-392-7724



Multimedia and Graphics

Listen to the Sounds: The MMPM/2 Audio Subsystem



Chris Dinallo

by Chris Dinallo

The Multimedia Presentation Manager/2 Audio Subsystem™ (MMPM/2) enables a PC to play and record audio waveform and MIDI data.

ANALOG VS. DIGITAL AUDIO

Analog audio, used in devices such as radios, tape decks, and microphones, is concerned with the generation, transmission, and reception of sound waves. These waves are characterized by amplitude, wavelength, and frequency. A sound wave's amplitude is measured by the amount, or volume, of atmospheric pressure it displaces. A wavelength is the distance sound travels during one complete cycle of pressure change, while frequency is the number of times a wave cycles per second. A high frequency (many

cycles) produces a richer, more audible sound. Figure 1 illustrates the characteristics of a sound wave.

In contrast to analog audio, digital audio is a study of discrete values. Digital audio records the same sound characteristics as analog audio, but catalogues each discrete moment with nonvarying precision. Digital computer techniques make it possible to generate, transmit, and receive digital audio with no loss of information. Because it is comprised of digital data, digital audio is not affected by outside factors, and therefore provides consistently repeatable sound reproduction. Compact discs (CDs) and CD players, the most common digital audio formats, bring high sound quality and absolute repeatability to the home stereo.

An alternate form of digital data that produces audio is called MIDI, or Musical Instrument Digital Interface. MIDI is digital protocol that is translated by synthesizers and musical instruments into waveform output.

With the technology of MMPM/2, PCs can now interact with digital and MIDI audio systems for recording and playback. The subsystem even has a built-in music synthesizer that enables the user to play MIDI data without MIDI instruments, providing a cost-effective entry point into the MIDI environment.

SOUND FLEXIBILITY

Hardware configurations for the Audio Subsystem can include an audio adapter (for example, IBM's M-Audio™), CD-ROMs, compact disc-extended architecture (CD/XA),

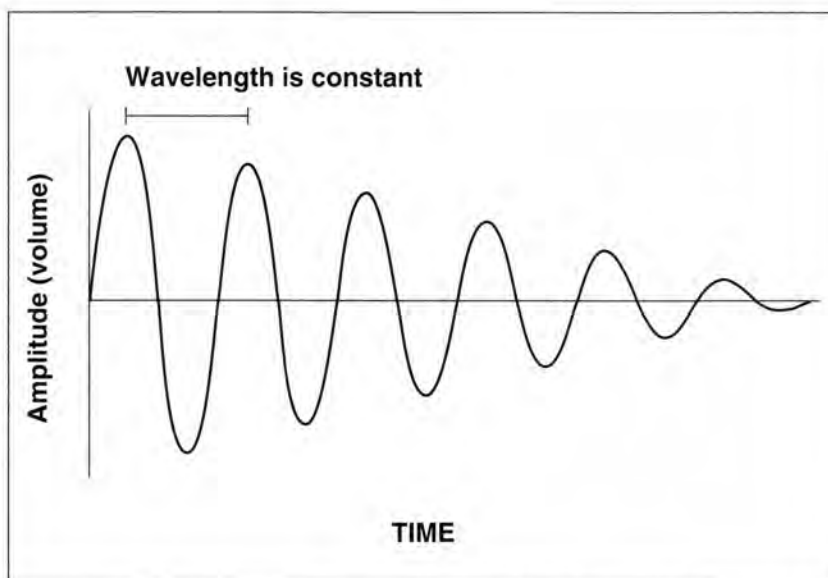


Figure 1: Sound wave characteristics. Amplitude decreases in time as wavelength remains constant.

and additional stereo equipment such as receivers and audio tape decks. One possible setup is shown in Figure 2.

The software for this setup is IBM's MMPM/2, an OS/2 2.0 multimedia subsystem made up of several code modules. Each module performs a specific function and can be replaced by custom-built third-party modules (such as the WaveAudio™ media control driver (MCD), MIDI sequencer MCD, Amp/Mixer MCD, Audio Stream Handler device driver, and M-Audio physical device driver) as long as these modules conform to MMPM/2 API interfaces.

A SOUND ARCHITECTURE

The MMPM/2 API interfaces consist of three layers: the Media Control Interface (MCI), the Streaming Protocol Interface (SPI), and the physical Device Drivers (PDDs). The three layers span OS/2's rings, with the media control drivers (MCDs) existing as ring 3 DLLs and the stream handlers existing as either ring 3 DLLs or (as in the Audio Stream Handler) ring 0 device drivers. This setup is shown in Figure 3.

A typical application would use the MCI to command the subsystem. Its functions include playback and record of audio data types and user controls such as volume, bass, and treble. The residing modules are DLLs (WaveAudio, MIDI Sequencer, and Amp/Mixer). In Figure 4, an application requests audio playback.

The SPI, primarily used by the Audio Subsystem's MCDs, controls data flow (which may consist of pulse code modulation (PCM), adaptive delta pulse code modulation (ADPCM), CD digital audio (CD-DA), CD/XA, and MIDI) to and from the audio device. The MCDs request data and specify when it is to be moved, request cue-points, and have event call-back routines. Cue-points, which act as place holders within the media, can be represented as time elapsed (for example, 30 seconds into a specific piece) or as a recurrent value (for example, every second throughout the piece). Cue-point notification is initiated during stream creation, with the MCD's registration of the event callback routine. Figure 5 illustrates how an application requests a cue-point.

On the receiving end of the SPI interface, stream handlers field SPI requests, control data flow, and monitor synchronization and cue-point detection. In Figure 6, an application or MCD initiates audio playback.

The third API level is the device drivers. The foundation of the audio subsystem, it allows communication from MCDs via standardized IOCTL calls, which instruct it to set up the

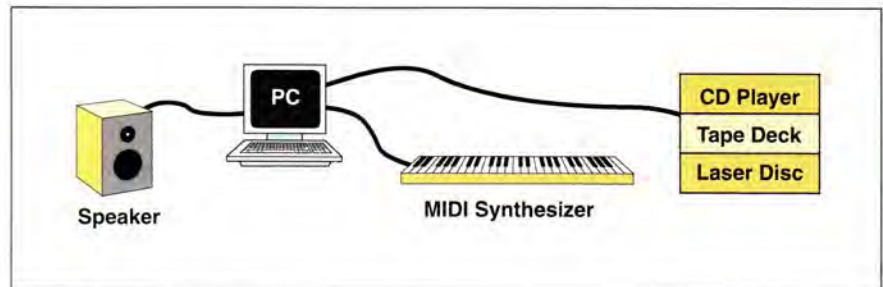


Figure 2: Multimedia hardware configuration options

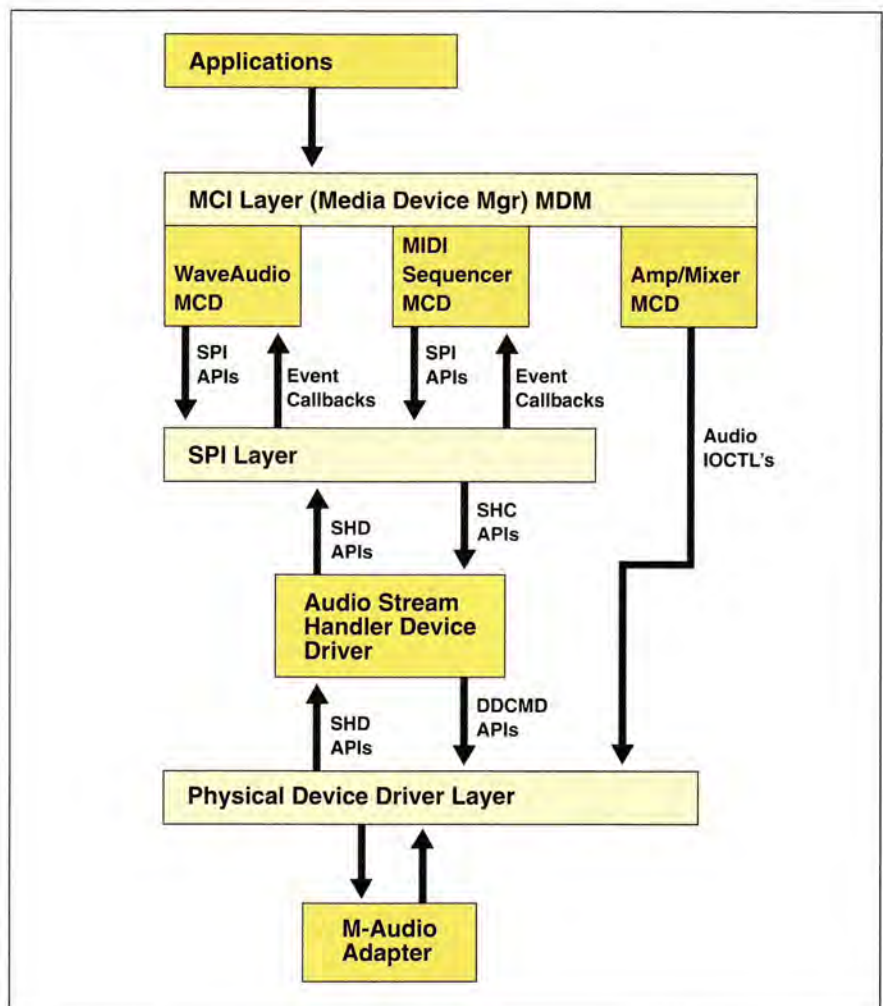


Figure 3: MMPM/2 Audio Subsystem modules within the three layers of APIs



```

MCI_OPEN_PARMS mciOpen;
MCI_OPEN_LOAD mciLoad;
MCI_PLAY_PARMS mciPlay;
DWORD          dwError;
HWND           hwndMyWindow;

mciOpen.dwCallback = NULL;
mciOpen.dwDeviceID = NULL;          /* this is returned */

mciOpen.lpstrDeviceType = (LPSTR)NULL; /* default connection*/
mciOpen.lpstrElementName = lpstrElementName;

dwError = mciSendCommand( (WORD) 0,
                          MCI_OPEN,
                          MCI_WAIT | MCI_OPEN_ELEMENT |
                          MCI_OPEN_SHAREABLE | MCI_READONLY,
                          (DWORD) &mciOpen,
                          UP_OPEN);

if (dwError)
    Abort(dwError);

mciLoad.dwCallback = (DWORD) NULL; /* We're waiting */
mciLoad.lpstrElementName = lpstrElementName;

dwError = mciSendCommand( mciOpen.dwDeviceID,
                          MCI_LOAD,
                          MCI_WAIT | MCI_READONLY,
                          (DWORD) &mciLoad,
                          (DWORD) NULL);

if (dwError)
    Abort(dwError);

mciPlay.dwCallback = (DWORD) hwndMyWindow;
dwError = mciSendCommand( mciOpen.dwDeviceID,
                          MCI_PLAY,
                          MCI_NOTIFY,
                          (DWORD) &mciPlay,
                          UP_PLAY);

```

Figure 4: Using MCI to open, load and play an audio file

audio adapter for specific requests. The device driver facilitates communication between the API and SPI, binding the “streaming” of audio data to the physical device driver and then to the audio adapter. The communication interface consists of device driver commands (DDCMD) and stream helper function (SHF) APIs. DDCMD APIs are used to make requests from SPI stream handlers to the physical device drivers, and SHD APIs are used by

physical device drivers to call back into the stream handlers. Figure 7 shows communication between the stream handler and the physical device driver.

A SOUND INVESTMENT

In any of the three layers of the API, hardware and software can be replaced by third-party modules. With this flexibility, the MMPM/2



```

/* Set a cuepoint at the 30 second mark in the media */

WORD                wDeviceID;
HWND                hwndMyWindow;
MCI_CUEPOINT_PARMS  CuePointParms

/* assign dwCallback the handle to the PM Window
   this will be used for returning MM_MCICUEPOINT messages */

CuePointParms.dwCallback = (DWORD)hwndMyWindow;
CuePointParms.dwCuepoint = (DWORD)90000; /* mmtime format */

mciSendCommand(wDeviceID,          /* device id */
               MCI_SET_CUEPOINT,    /* MCI message */
               MCI_SET_CUEPOINT_ON, /* flags */
               (DWORD) &CuePointParms, /* data struct */
               0);                  /* no user parms*/

```

Figure 5: Using MCI to set cuepoints

```

spcbkey.ulDataType = DATATYPE_WAVEFORM;
spcbkey.ulDataSubType = WAVE_FORMAT_2S08; /* 22kHz 8bit stereo */
spcbkey.ulIntKey = 0; /* no internal key */
strcpy(dcb.szDevName, AUDIO01$ ); /* device name to use*/
dcb.ulSysFileNum = ulSysFileNum; /* from AUDIO_INIT IOCTL
pdcBTgt = (PDCB)&dcb;

SpiCreateStream(hidSource,          /* Source handler id */
               hidTarget,          /* Target handler id */
               &spcbkey,          /* stream protocol */
               NULL,              /* Src DCB */
               pdcBTgt,          /* Tgt DCB */
               &evcb,            /* event control blk */
               (PEVFN) MyEventRtn, /* addr of event rtn */
               0,                 /* not a split stream*/
               &hStream,          /* hndl returned */
               &hEvent);          /* hndl returned */

/* Use MMIO to get audio file and fill in ACB structure */
/* Next, make association of audio file to stream created */

SpiAssociate(hStream, hidSource, (PACB)&acb);

/* Start playback of stream by issuing SpiStartStream() */

SpiStartStream(hStream, SPI_START_STREAM);

/* Audio will play until end of stream, at which time
   the registered event routine will be called. */

```

Figure 6: Using SPI to create an audio stream for playback



```

DDCMDREGISTER  DDCMDRegister;
PDCB_AUDIOISH  pDCB;

/* Stream Handler to register a stream with the physical DD. */
/* Get device info and attach to the audio PDD via DevHlp */

if (DevHlp_AttachDD(nszAudioDDName, &AttachAudio)) {
    return(ERROR_DEVICE_NOT_FOUND);
}

/* make a 16:16 pointer to call PDD */

pSTREAM->pDDCMDEntryPoint = (PDDCMDFN)MAKEP(AttachAudio.protCS,
                                           AttachAudio.protOFF);

DDCMDRegister.ulFunction = DDCMD_REG_STREAM;
DDCMDRegister.hStream = pSTREAM->hStream;
DDCMDRegister.ulSysFileNum = pDCB->ulSysFileNum;
DDCMDRegister.pSHDEntryPoint = &SHDEntryPoint;
DDCMDRegister.ulStreamOperation = STREAM_OPERATION_PRODUCE;
DDCMDRegister.spcbkey.ulDataType = pCreate->spcbkey.ulDataType;
DDCMDRegister.spcbkey.ulDataSubType = Create->spcbkey.ulDataSubType;
DDCMDRegister.ulBufSize = 0;
DDCMDRegister.ulNumBufs = 0;
DDCMDRegister.ulBytesPerUnit = 0;
DDCMDRegister.mmtimePerUnit = 0;
DDCMDRegister.ulAddressType = ADDRESS_TYPE_PHYSICAL;
rc = (*pSTREAM->pDDCMDEntryPoint)(&DDCMDRegister);

/* Physical device driver calling Stream Handler during interrupt time */

SHD_REPORTINT  ShdInt ;

ShdInt.ulFunction = SHD_REPORT_INT ;    /* report interrupt */
ShdInt.hStream    = pStream->hStream;    /* stream hndl */
ShdInt.ulFlag     = SHD_WRITE_COMPLETE; /* function done */
ShdInt.ulStatus   = ulBytesPlayed;      /* # of bytes written*/
ShdInt.ulStreamTime = ulIncrementalTime; /* millsec time */
((PSHDFN)pStream->ADSHEntry)(&ShdInt); /* IDC call to SH */

```

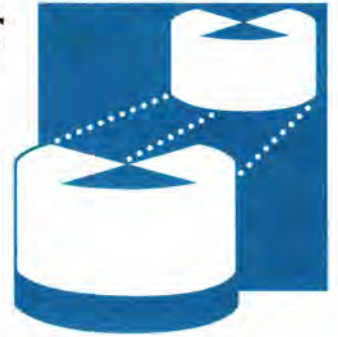
Figure 7: Using DDCMDs and SHDs for Inter-Device Driver Communication (IDC)

architecture will be able to incorporate future advances in technology and protect customer investments well into the future.

MMPM/2's ability to process many sounds at various quality levels while providing a flexible hardware and software environment can transform a PC into a personal multimedia entertainment center or recording studio with much potential for growth.

Chris Dinallo, IBM Corp., Personal Systems Programming, P.O. Box 1328 Boca Raton, Fla., 33429. Dinallo is a staff programmer in the Multimedia Software Development department and is the lead developer of the MMPM/2 Audio Subsystem. He has worked in Multimedia Software since its inception in March 1990. His prior experience includes firmware and DOS kernel programming. Dinallo has a B.S. in computer science from the University of Florida.

Database Manager



Advanced Programming Considerations for Database Manager

by Lance Amundsen and Richard Hoffman

Database design and development are driven by factors such as normalization, integrity constraints, and DASD requirements. Programming guidelines are often affected by concerns about portability or by the need to maximize use of existing code. On the other hand, performance and ease of use are prime concerns for those who must use a database application day to day, long after design and coding is done.

This article focuses on features unique to Database Manager that can be used to enhance application performance and improve application usability. These features include locking options, static vs. dynamic SQL usage, and record blocking.

LOCKING OPTIONS

Most databases accessed by multiple users have some method of locking to avoid modification anomalies. Locking, however, can decrease concurrency, the ability of multiple users to efficiently update the database at the same time. Locking can also degrade performance, especially when locking "granularity" is increased.

Figure 1 shows the tradeoff between performance and concurrency as a function of locking granularity.

ISOLATION LEVELS

Database Manager uses the isolation level option to determine the basic locking scheme for cursors in an application. Three isolation levels are offered for Database Manager applications.

Repeatable Read

As they are fetched for a cursor, rows are locked and remain locked until the end of a transaction, thus producing a "repeatable read" cursor. For example, as rows are fetched for a cursor that references "all employees with salaries greater than \$30,000," each row is locked. Repeatable read ensures that the next time the cursor is opened and read within the same transaction, those rows of the cursor will be unchanged by other applications or processes.

Row-level locking under repeatable-read isolation is supported only if the data is accessed with an index. If a table scan is used, the entire table is then locked.

Cursor Stability

The currently accessed row of a cursor and previously changed rows in the same transaction are locked. When a cursor is processed using FETCH statements, cursor stability ensures that any fetched row is locked, until the next FETCH statement is processed. This allows an application to retrieve, examine, and change a single row without another application or process modifying that row.

Uncommitted Read

No row locks are acquired or examined. This isolation level is used for cursors that read data without updates. However, because uncommitted read cursors do not examine locks, it is possible to read the uncommitted data of other applications. This uncommitted data might not be valid, because another application's transaction could eventually be rolled back instead of committed.



Lance Amundsen



Richard Hoffman



Because uncommitted read, or "dirty read," cursors cannot be used to update a database, this isolation level applies only to the "read-only" cursors in an application. Cursors that can be updated function as if the cursor stability isolation level were in effect.

LOCK TABLE STATEMENT

If an application requires more explicit control over locking, the SQL statement **LOCK TABLE** can be used to lock an entire table in either **EXCLUSIVE** or **SHARED** mode.

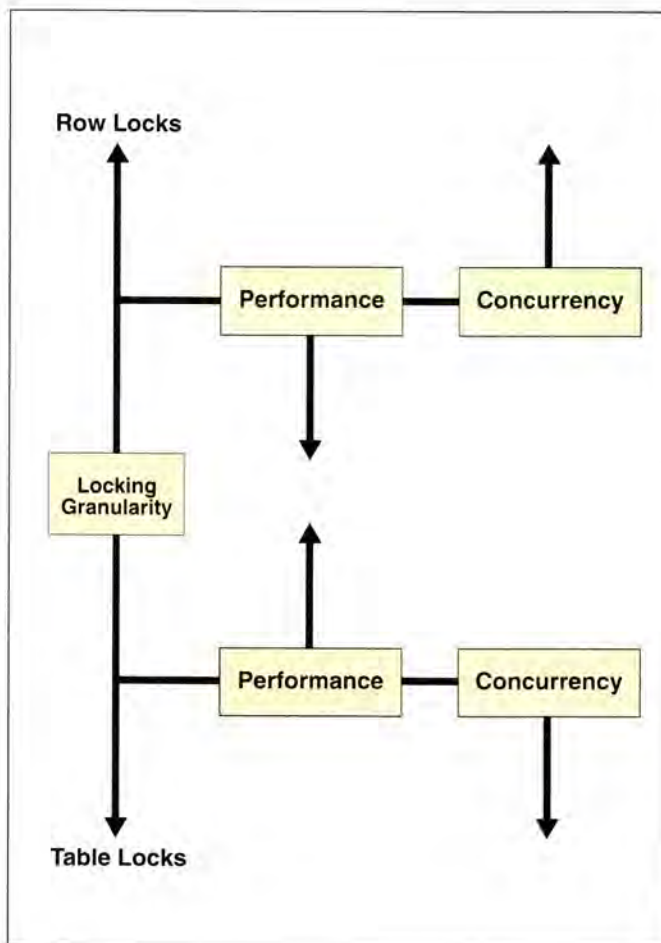


Figure 1: Performance and concurrency variations due to changes in locking granularity

SHARED mode allows other applications to read data in the locked table, but not to modify that data, while **EXCLUSIVE** mode prohibits updates and reads by other applications, except for those using the uncommitted-read isolation level.

DATABASE CONFIGURATION PARAMETERS RELATED TO LOCKING

Parameters in the database configuration file associated with each database can affect locking performance. The two most important are **LOCKLIST** and **MAXLOCKS**.

Locklist

The size of the lock list Database Manager stores in memory for all applications accessing the database. This value is specified in 4K pages, with between 80 and 160 locks allowed per page. The range of values allowed is 4 to 250 with a default of 8.

Maxlocks

Maxlocks refers to the percentage of **LOCKLIST** an application can use before escalation. Escalation is the process of converting the row locks of the application's most locked table to an explicit table lock.

With escalation, concurrency decreases and the benefits of row-level locking start to disappear. It is therefore important to minimize escalation by choosing a large enough value for **LOCKLIST** (25 is a good starting point) and by carefully controlling **COMMIT** points within the application to ensure that it acquires no more locks than allowed by the **MAXLOCKS** percentage.

LOCKING TIPS AND STRATEGIES

Repeatable-read applications accessing a large percentage of the rows in a table can often benefit by a **LOCK TABLE** statement. The overhead of individual row locks is eliminated with, in some cases, only a minor reduction in concurrency.

Table locks can be automatically acquired by Database Manager when lock escalation occurs or when a table scan is used to access a repeatable read cursor. To avoid unwarranted table locks and increase concurrency, avoid escalation and access repeatable read cursors via an index. The Database Manager **EXPLAIN** utility can be especially helpful for determining the access method used by Database Manager.

When not updating data, consider the uncommitted-read option. Although this maximizes performance and concurrency, you must remember the implications of reading uncommitted data.

Don't overlook the uncommitted read option when updating data, especially if you make infrequent changes. It is possible to read data and then issue an explicit searched update for a record. A unique field such as a timestamp is especially helpful in this situation. Even without one, the **WHERE** clause of the **UPDATE** statement could contain sufficient information to guarantee that another application hasn't changed the record since it was last read. Without a timestamp, however, the entire record might have to be specified in the **WHERE** clause, and duplicate rows could present a problem.

STATIC VS. DYNAMIC SQL USAGE

Database Manager provides both static and dynamic SQL. Static SQL increases performance while dynamic SQL allows greater flexibility. Figure 2 shows the major differences between static and dynamic SQL.

DIFFERENCES BETWEEN STATIC AND DYNAMIC SQL

Preparation

The preparation of an SQL statement involves parsing and syntactic and semantic checking of the statement text, followed by an optimization step that determines how to efficiently access data. An executable form of the statement is then created.

For static SQL, this step is performed at precompile time and optionally repeated at later bind times. The executable form is stored in a package, or access plan, in the database. One restriction of static SQL is that complete statement syntax must be specified at precompile time. Host variables can substitute values at run time, but the statement must reference only existing objects.

For dynamic SQL, this step is performed on the fly, while the statement is executed by the user. One advantage of this setup is that the

SQL statement text can be generated at run time. Preparation time can be significant, however, and can hurt performance.

Persistence

The persistence of an SQL statement is the life of the executable form of the statement, or the length of time between a statement's preparation and the need for subsequent preparation.

For static SQL, the executable form lasts as long as the database package that contains it lasts, or until a new executable form replaces it during package regeneration, such as when the application is rebound.

For dynamic SQL, the executable forms lasts only as long as the transaction. A dynamic SQL statement must be reprepared after a **COMMIT** or **ROLLBACK**. The one exception to this rule is for cursors declared **WITH HOLD**. Unlike a **ROLLBACK**, a **COMMIT** does not destroy a dynamic **WITH HOLD** cursor or any dynamic statements associated with it.



Item	Static SQL	Dynamic SQL
Preparation	At application development time	At run time
Persistence	Until rebound	For the duration of the transaction
Authorization	Binder	User
Optimization Assumptions	Those present at bind time	Current

Figure 2: Differences between static and dynamic SQL

Authorization

The authorizations required to process an SQL statement can be checked either at application bind time or at run time, with the fundamental difference being the authorities of the binder and user.

For static SQL, the authorization of the binder is used for all data manipulation SQL statements, but data definition statements use



The total time it takes to run a query is $T = P + E$, where P represents preparation time and E represents execution time.

the authorization of the user. Users need only the `EXECUTE` privilege on the package to invoke any of the DML SQL statements in the program. One advantage to this is that a user can be restricted from full `UPDATE` privileges on a table while still being allowed to update that table using a limited number of static SQL statements contained in the application.

For dynamic SQL, the user authorization is checked for all SQL statements. Each user must therefore have the required privileges to execute a SQL statement dynamically.

Optimization Assumptions

Optimization uses database statistics to determine an efficient way to access data.

For static SQL, optimization uses the statistics present when the application is bound. These statistics, however, might or might not be an accurate representation of those that exist when the application is actually run. It is therefore important to periodically use the `RUN STATISTICS` function followed by rebinding all static SQL programs, especially when there are significant changes to the database.

For dynamic SQL, the most recent statistics are always used. This can dramatically affect performance in some cases, such as when an index is added that could be taken advantage of by a dynamic query. The same static query couldn't incorporate this information without being rebound.

THE BOTTOM LINE

So what does all this mean? One thing to consider is that the total time it takes to run a query is $T = P + E$, where P represents preparation time and E represents execution time.

If a query is complex but doesn't return many rows, P might be large compared to E . In this case, a static version of the SQL statement would run more quickly since the preparation time is performed at application development time instead of at run time.

If a query isn't complex and returns many rows, P might be small compared to E . Here, a dynamic version of the SQL statement could

run more quickly since more current optimization assumptions are used when the statement is prepared at run time.

A further consideration is that certain assumptions can be made for some dynamic SQL statements that can't be made for similar static SQL statements using host variables. One case of this is a dynamic query using a literal constant in a `WHERE` clause, instead of a static query that substitutes the value in a host variable at run time.

For example, a complex query involving many joins containing a `WHERE` `COL` greater than 100 could be more efficiently optimized when the database statistics indicate that only a small number of rows, or no rows at all, satisfy the criteria. The join order would be performed to eliminate as many rows as possible as soon as possible. This information is not available to the optimizer, however, when the value in the `WHERE` clause is provided by a host variable at run time.

Other examples can be found in which a dynamic form of a SQL statement executes much faster than a similar static statement. This is especially true for a `WHERE` clause containing a `LIKE` "string literal" without a wildcard as the first character. An index on the column could be used, but not if a host variable substituted the value at run time.

A final point to consider is the persistence of dynamic SQL statements. Every application must periodically issue `COMMIT` statements to maintain high concurrency and force physical writes of the database logs. This helps to avoid data loss in the event of a power failure. Dynamic statements must be reprepared after a `COMMIT`, which can take considerable time if done repetitively, such as in tight program loops.

RECORD BLOCKING

Consider the following scenario: an application displays 30 records at a time to a user browsing the data. The data is located on a server on a busy network; response time can be as high as half a second. Each time the user pages down, the application issues another 30 `FETCH` statements, each of which taking up to half a second to process remotely. The total

time elapsed between screens of information could be as high as 15 seconds.

Although hypothetical, this example points out that network transfer time is not negligible and can make up a significant percentage of total processing time.

A faster approach might be to return a block of rows. The subsequent `FETCH`s would occur locally, reducing network traffic from 30 requests to as few as one. This is where record blocking comes in: Database Manager allows read-only cursors to be blocked, as in Figure 3.

With record blocking in effect, the first `FETCH` for a cursor causes a block of rows to be sent to the client. Subsequent `FETCH`s retrieve data from a local buffer in which Database Manager has automatically allocated and stored the block of rows. When all rows from the block are retrieved, another block is sent to the client.

All this activity is transparent to the application except for the increased performance and the difference between the "virtual" cursor position and the "actual" cursor position. On the first `FETCH`, the application is positioned on the first row. The cursor position at the server, however, is different. The actual cursor is located on the row first returned with the next block of rows.

This situation leads to the restriction that blocked cursors cannot be updated. This is because the row an application is examining is not the "actual" cursor position and is therefore unlocked. Other applications might have modified the row between the time the block of rows was sent to the client and the time the client actually fetched the row.

BLOCKING OPTIONS AND AMBIGUOUS CURSORS

Record blocking is automatically enabled by a precompile and bind option. The `/K` option can be specified as follows:

- `/K=NO`—Do not block any cursors
- `/K=UNAMBIG`—Block all cursors known to be read only
- `/K=ALL`—Block cursors that are known to be read only and cursors that are ambiguous

To define ambiguous cursors, consider that updates to a cursor cannot occur if the cursor is based on a read-only `SELECT` statement, such as one referencing a nonupdatable view or involving a `UNION`.

Cursor updates cannot occur if the cursor involves a sort, as with a `SELECT` statement involving a `DISTINCT`, `ORDER BY`, or `GROUP BY` clause, or if the cursor is declared `FOR FETCH ONLY`.

On the other hand, Database Manager assumes that updates will occur if the cursor is declared with a `FOR UPDATE` clause, or if a `DELETE` or `UPDATE WHERE CURRENT OF` statement is associated with the cursor.

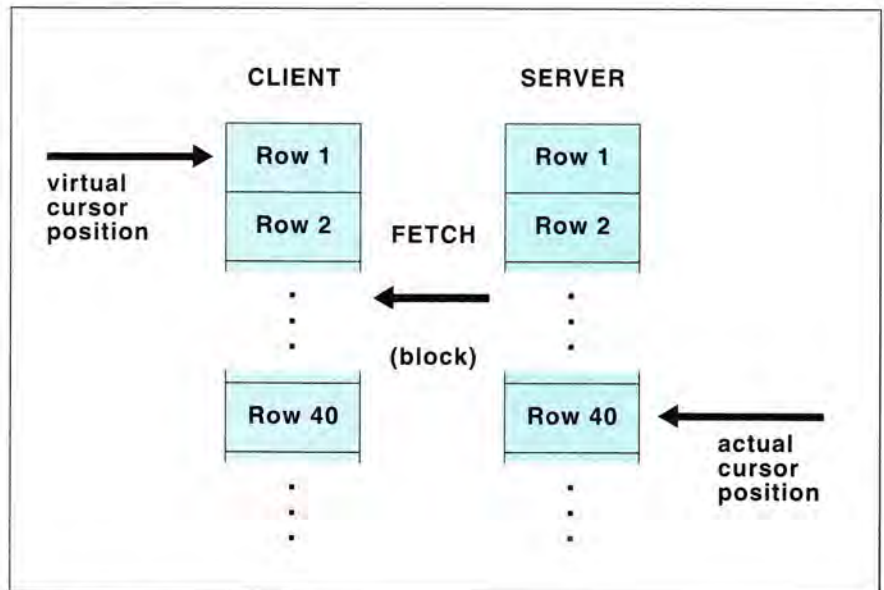


Figure 3: Blocking read-only cursors

If neither of the previous situations is true, the cursor is assumed to be read only unless a `PREPARE`, `EXECUTE`, or `EXECUTE IMMEDIATE` statement appears in the same source file of the application. In this situation, a dynamic SQL statement that updates the cursor, such as `DELETE WHERE CURRENT OF`, could be executed. This is called an ambiguous cursor.

Ambiguous cursors are not blocked if the default blocking option, `/K=UNAMBIG`, is used. They are blocked, however, if the `/K=ALL` option is used. Since data integrity is so important, any update attempt against a blocked cursor will return an error.



BLOCKING PARAMETERS

The following Database Manager Configuration File parameters affect blocking performance:

- **SVRIOBLK** indicates the maximum size of the block of rows the server can send to the client when a blocked cursor is fetched. This value ranges from 4K to 64K.
- **RQRIOBLK** indicates the size of the block of rows an OS/2 database client requests when a blocked cursor is fetched. This value ranges from 4K to 32K, with a default of 4K.

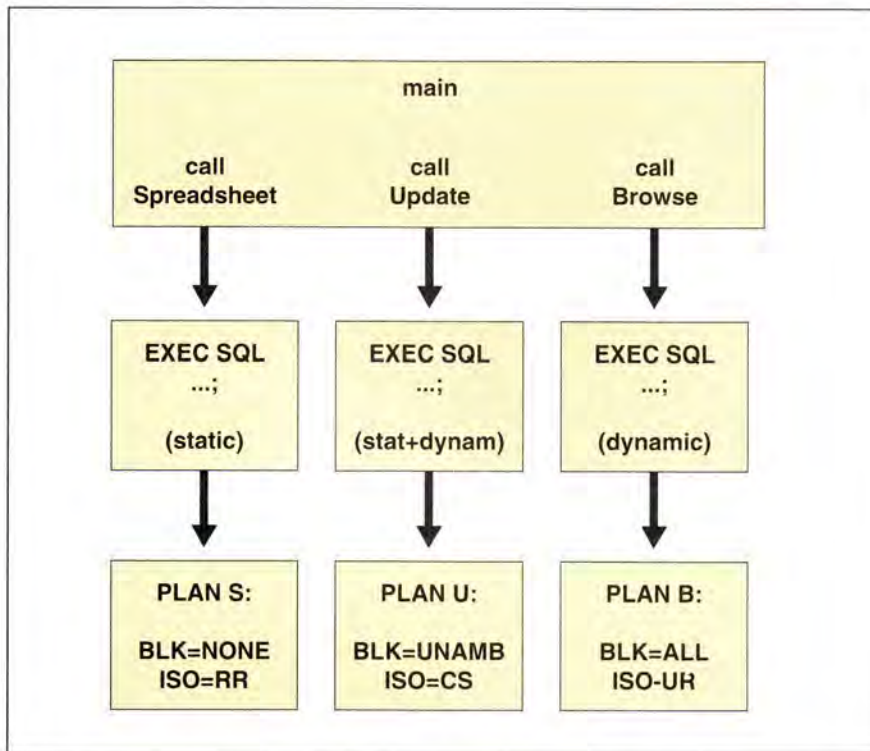


Figure 4: Application using multiple packages, isolation levels, and blocking options

- **SQLSIZE** indicates the size of the block of rows that DOS and Windows database clients request when a blocked cursor is fetched. This value ranges from 512B to 64K, with a default of 1K.
- **COMHEAPSZ** gives the size of the client communications heap for OS/2 database clients in 64K shared segments. Set this value to at least $1 + m \times RQRIOBLK / 64KB$, where m is the number of open cursors for the application.

- **NUMRC** indicates the number of communication heaps available at the server. If this value is exceeded, blocking is disabled for OS/2 database clients and a connection attempt is refused for DOS and Windows Database clients.

MAXIMIZING RECORD BLOCKING

To maximize record blocking,

- Declare all read-only cursors as **FOR FETCH ONLY**. This completely eliminates ambiguity.
- With high-volume, often-run queries, choose **SVRIOBLK** and **RQRIOBLK** to be as large a multiple of the row size as possible.
- Do not mix static and dynamic SQL in the same source file unless ambiguous cursors can be tolerated (by using the **/K=ALL** option, for example).
- Finally, bind or precompile with **/K=ALL**, unless updates or "true" cursor stability are required.

GOING ONE STEP FURTHER

The record blocking feature of Database Manager does have some limitations. The parameters affecting record blocking are system wide, and the maximum amount of data that can be blocked is 32K for OS/2 Database clients and 64K for DOS and Windows database clients.

Fortunately, use of the Database Application Remote Interface feature (ARI) of Database Manager allows an application to be split into a client procedure running on the client and a server procedure stored on the database server. The server procedure runs at the database location and can return information, such as a block of rows, to the client. Server procedures can be tailored to the specific blocking needs of the application and are not subject to 32K and 64K blocking limits.

Using ARI for custom record blocking is beyond the scope of this article, however; additional information can be found in the References box.

PUTTING IT ALL TOGETHER

This article has covered a broad range of topics that can be combined in the same application. A single application can access multiple packages, each including static or dynamic SQL and with each package bound with different isolation levels and blocking options. Figure 4 shows how a single application can handle the needs of various application types.

Spreadsheet

This application displays more than a single row of information while allowing a user to browse and edit the data. The spreadsheet lends itself to static SQL, with a repeatable read isolation level to ensure that the all data examined by the user is locked. No record blocking is used, as updates are allowed.

Single Record Update

This application involves a single record retrieval based upon some criteria specified by the user, such as a name or employee ID. The user is then allowed to update the record. A combination of static SQL and the cursor stability isolation level works for this application; no blocking is used. A more complicated version would allow a user to enter an arbitrary query, defining a set of rows that could be examined and updated one at a time. Dynamic SQL works best here.

Browse Only

This application allows a user to specify an arbitrary query and to view but not change the resulting data. A dynamic SQL approach, with the uncommitted read isolation level and record blocking, gives optimal performance.

Lance Amundsen, IBM Corp., 11400 Burnet Rd., Austin, Texas, 78758. Amundsen is currently lead programmer for the Database Manager Windows Database Client Application Enabler. He joined IBM in 1989 as the author of the Database Manager Programming Guide and Reference. He holds a B.S. in physics from the University of Minnesota.

Richard Hoffman, IBM Corp., 11400 Burnet Rd., Austin, Texas, 78758. Hoffman works as IBM's technical liaison to Taligent, an IBM/Apple joint venture. He joined IBM in 1988 as SQL architect for Database Manager. He holds a Ph.D. in mathematics from the University of Texas at Dallas.



REFERENCES

IBM Extended Services for OS/2 Database Manager Programming Guide and Reference (IBM Doc. S04G1022) contains descriptions of locking, isolation levels, static and dynamic SQL, record blocking, and ARI.

Parameter and Tuning Guide for OS/2 EE (IBM Doc. G33F9437) contains information on adjusting Database Manager and Communications Manager parameters for better performance.

Amundsen, Lance and Nameeta Sappal. "Passing Blocks of Data Using ARI," *IBM Personal Systems Developer* (Spring 1991): 98-103.

Jacobs, Dwayne. "You Too Can Feel the Power," *IBM Personal Systems Developer* (Spring 1989): 92-97.

Malkemus, Tim. "Database Manager Optimizer," *IBM Personal Systems Developer* (Summer 1989): 74-80.



Database Manager

User Exits: Using Nonstandard Devices with DBM

You Can Back Up Your Database to Tape

by Dawn E. Bezviner and Richard Lawrence

The User Exit is a new feature of the OS/2 Extended Services 1.0 Database Manager (DBM) that allows you to read and write to devices other than those supported by OS/2. With a User Exit, a database and its log files can be backed up to a nonstandard device (NSD) such as a tape unit, optical disk, or host machine. This article describes how the User Exit benefits the database user and how it is used by DBM, as well as offering some tips for writing one.

There is a difference between a physical device and the software that drives it. Throughout this article each will be referred to as an NSD, since it should be clear from the context which is intended. Not all NSDs will be suitable for calling. Some advice for selecting one is included in this article. For a full description of the criteria for selecting an NSD, see IBM Extended Services for OS/2 Guide to Database Manager.

A User Exit is a user-supplied program that acts as a go-between for DBM and an NSD. Since the NSD is not supported by the underlying operating system, DBM cannot interact with it directly. Instead, it calls the User Exit, which in turn interacts with the NSD. The User Exit then returns to DBM, indicating whether or not an operation was a success.

A User Exit can be called from two functions. The first is the DBM **Backup and Restore** function. DBM users had had a requirement to back up and restore databases to tape devices. Backing up a large database to disk is time-consuming and error-prone because of the large number of disks involved. Yet since DBM uses the OS/2 **Backup and Restore** functions, users were restricted to devices supported by OS/2: disks, hard disks, and LANs. Now the DBM **Backup and Restore** functions can be directed to call a User Exit, enabling the user to connect to an NSD.

The second function that can call a User Exit is roll-forward recovery. Roll-forward recovery is a new feature that allows a database to be restored from a backup and brought up to a specific point in time by replaying the log files since that backup. An NSD is a logical place to store these log files.

A CLOSER LOOK AT A USER EXIT

Figure 1 shows how the DBM, User Exit, and NSD interact. The DBM does not interact directly with the NSD at all; hence, neither DBM nor the database application needs to support the NSD. Several software products are available that support **Backup and Restore** functions between OS/2 and an NSD. When setting up a system, the database

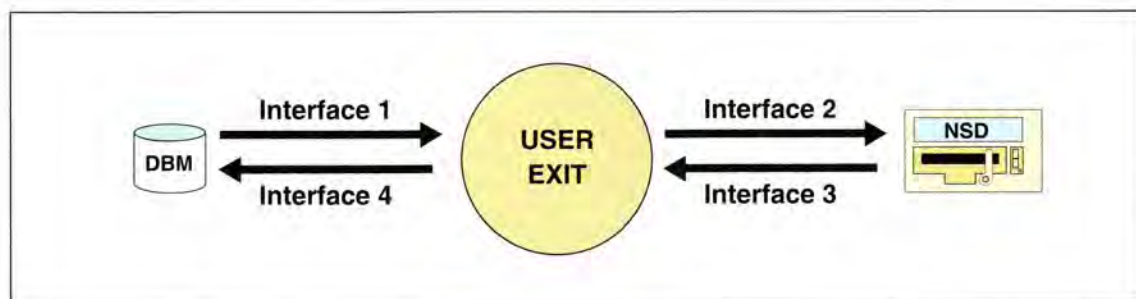


Figure 1: Interaction between DBM, User Exit, and NSD



administrator chooses one, along with one of the NSDs it supports. Then the application writer writes a User Exit that interacts with DBM and the NSD.

Let's look at the interfaces in Figure 1. First, we see the DBM calling the User Exit. DBM passes several different parameters, depending on which function is being called. The User Exit may support up to four functions:

- Back up a database
- Restore a database
- Back up (or archive) a log file
- Restore (or retrieve) a log file.

The first parameter, **Backup**, **Restore**, **Archive**, or **Retrieve**, indicates which function is being called. (The functions will usually need to be written in pairs, **Backup** with **Restore** and **Archive** with **Retrieve**.) The other parameters specify the database to back up, the log file to archive, and the path of the log file. We will discuss these parameters later. (They are also described in depth in *IBM Extended Services for OS/2 Guide to Database Manager*.)

The User Exit was designed to be flexible, and you should customize it according to the requirements of each database. For example, you might want to back up one database to an NSD and a second database to a different NSD. This tactic would also require that when the **Restore** function is called, you determine from which database to retrieve the information. To reduce wear on your tape device, try to archive several files at a time. Another good idea for handling log files is to compress them when archiving and decompress them when retrieving. For example, PKZIP™ is a product, shipped with Extended Services 1.0, that compresses and decompresses files. A database installation might have two NSDs, one used only when the first is down or unusable. In this case, the User Exit must determine which NSD to use.

The second interface in Figure 1 shows the User Exit interacting with the NSD. Some extra work must be done here by the User Exit, as different **Backup** and **Restore** products will want information passed to them in different ways. For example, the Mountain Networking Solutions™ tape software allows all

information to be passed by either command line or previously written procedure. The Sytron SytosPlus™ tape software, on the other hand, requires most of the information to be specified in a previously written procedure and allows only limited changes through the command line. The User Exit must know the name of the procedure; DBM will not. The User Exit must translate all information from the form in which the DBM passes it to the form the particular product being called requires. Then, the User Exit does the actual invocation.

RETURN CODE	MEANING
0	Successful
4	A resource error occurred
8	Operator intervention is required (for example, drive door is open)
12	Hardware error occurred
16	Defect, configuration, or installation error occurred
20	Invalid parameter received
24	Required files not found
28	Unknown error occurred
32	Operator cancelled the function

Table 1: Return codes

The third interface shows the NSD returning to the User Exit. In general, this will be a return code indicating whether or not the task was completed successfully. At this point, the User Exit can return to DBM, or it can do further processing and reinvoke an NSD. In the example where the database installation has two NSDs, one for use when the first is down, the User Exit should determine which NSD was called. If it was the first, the User Exit should do whatever processing is needed to call the second NSD and invoke it.



The fourth interface shows the User Exit returning to the DBM. DBM will handle any of nine return codes. Since the NSD is not likely to have the same set of return codes, the User Exit must map the NSD's return codes to an appropriate DBM return code. The nine return codes are shown in Table 1.

The DBM uses the different return codes in two ways. The first way is to determine whether the operation was successful. (If a log

computer when it is running. Some database users might not have an operator on hand, for example, for running an overnight backup. If an operator is present, Figure 3 shows how he or she can interact with the User Exit.

The operator generally does *not* interact directly with DBM. Even when the DBM uses the OS/2 Backup or Restore function, it is that function, not the DBM, that prompts for drives that aren't ready, prompts for the next disk, or checks and prompts when the wrong disk is inserted. When a User Exit is in use, either it or the NSD driver has to do that prompting. Specific error conditions are discussed later.

EXAMPLES

We have mentioned several ways in which to customize the User Exit. This section will give several examples of how to code some common functions. The application writer looking for more complete examples can also look at the four sample User Exits shipped with Extended Services 1.0 DBM.

Figure 4 shows how a User Exit might direct different databases to different NSDs. Suppose one database is very large, and the second is fairly small. In the example the large database, called DBONE, is backed up to a tape device using the Mountain Networking Solutions tape software. The second database, called DBTWO, is compressed and backed up to a hard disk using PKZIP.

Figure 5 shows how a User Exit might group log files before archiving them. Sytron SyPlus allows only a limited amount of information to be passed via the command line, with the rest specified in a previously saved menu-driven procedure. In this example, the application writer has created and saved a procedure called "Archlogs" which writes all files in the COLLECT.LOG directory to the NSD.

Here, the User Exit collects up to five log files before calling the NSD. Each time the User Exit is called to archive a log file, it first copies it to another directory, called COLLECT.LOG, compressing the file as it does so. If the number of the log file is divisible by 5, the User Exit then calls the NSD to write out the files. If successful, it erases the files from the

```

/***** MAPRC PROCEDURE *****/
/* Check the return code from the MaynStream Tape Backup Software */
/* and it to the expected DBM return codes. */
/*****/
maprc:

Select
  when rc = 0 then do                                /* Successful */
    sqluexitrc = IBM_RC.OK;
    return;
  end
  OTHERWISE do                                       /* UNKNOWN ERROR */
    sqluexitrc = IBM_RC.UNKNOWN;
    return;
  end
end

return

```

Figure 2: Mapping return codes

file was not archived successfully, the DBM will not erase it.) The second way is to let DBM decide whether it can continue making calls to the User Exit. For example, if operator intervention is required, then a future call might succeed. However, if a hardware error occurred, future calls will not succeed. Since the DBM should always pass valid parameters, a return code of 20 indicates a serious internal problem (or a bug in the User Exit's parameter checking). As an example, Figure 2 shows a sample of how return codes would be mapped for the Maynard Maynstream tape software. The User Exit should also clean up any temporary files or directories it created before returning to the DBM.

So far, it has been assumed that the User Exit is written as though no operator is present at the

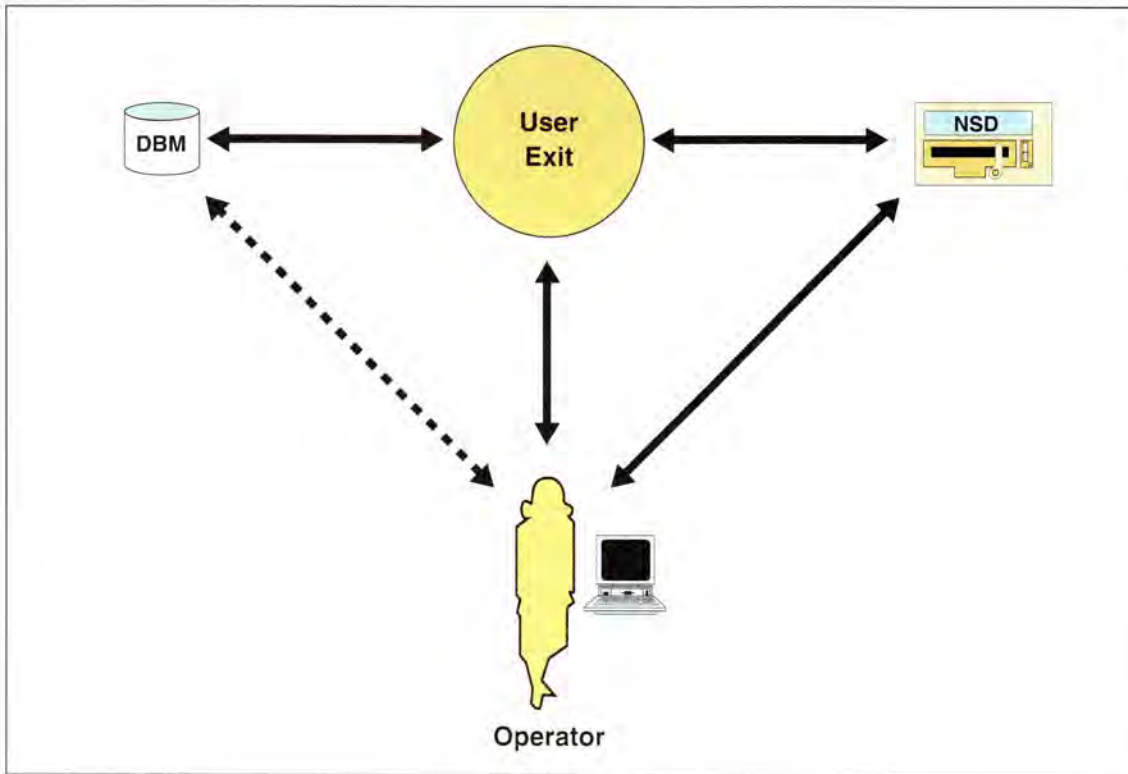


Figure 3: Where the user fits in

COLLECT.LOG directory and returns a successful return code. However, if the log file number is not divisible by 5, the User Exit simply sends a successful return code to the DBM, without invoking the NSD.

There is a risk involved in saving log files before writing them to the NSD. If the database crashes, it can be recovered only to the latest point in the saved log files. The User Exit can, and should, check for log files in the COLLECT.LOG directory if they are not retrievable from the NSD, but if the disk containing the COLLECT.LOG directory is lost, then so are those log files. Any database requiring up-to-the-minute recovery, then, should either not save log files in this way or should save them on a different disk from that containing the database—not just on a different portion of the same disk, but on a physically different disk. That way, if either the database disk or the log file disk is lost, the other still exists as a backup.

Figure 6 shows how the User Exit might prompt an operator in order to request operator intervention on the NSD.

SUPPORTED LANGUAGES

All of the examples shown so far have been written in OS/2 REXX. The User Exit can be written in any language that is supported by OS/2, allows a variable number of parameters to be passed, and can handle the characters appropriately (for example, the backslash in the path). The DBM will call `SQLUEXIT parm1 parm2 ...`, and OS/2 will search the `PATH` for an executable file with that name. The extension could be `.CMD` or `.EXE`. Figures 7 and 8 show code fragments for writing the User Exit in C.

To create a User Exit using C, the application writer simply treats the parameters passed in from the DBM as `NULL` terminated strings. Since the parameters are passed to the OS/2 command processor as command line arguments, each is referable via the built-in C pointer array `argv[ ]`.

Table 2 shows how the parameters relate to the `argv[ ]` array for a specific `Backup` call.

Figure 7 shows entry into the User Exit. It logs the parameters and then calls a subroutine, `Backup`, `Restore`, `Archive`, or `Retrieve`, depending on the function requested by DBM.

The User Exit can be written in any language supported by OS/2 that allows a variable number of parameters to be passed and can handle the characters appropriately.



BACKUP_PROC:

```

/* NOTE: this routine assumes that the input parameters to the */
/*       User Exit were parsed as follows:                      */
/*       parse upper arg OPT PARM1 PARM2 PARM3 PARM4 PARM5      */

DB_DRIVE = strip(PARM1)      /* ex. D:                      */
DB_NAME  = strip(PARM2)      /* ex. DBONE                  */
RESP_FILE = strip(PARM3)     /* ex. D:\SQLDBDIR\SQL00001.BKP */
LABEL    = strip(PARM4)     /* ex. DBONE-684686581        */
MODE     = strip(PARM5)     /* ex. 1                      */

if( DB_NAME = 'DBONE' ) then
do
  /* Strip timestamp from the generic LABEL */
  parse var LABEL . '-' dbm_time
  /* Create a label for Mountain */
  mtn_label = 'DATABASE DBONE BACKUP : PART 'MODE,
              ' : TIMESTAMP = 'dbm_time

  /* Create the command line for Mountain, and put it into */
  /* the response file                                     */
  ctl_line = ' /L'mtn_label' /A/D/C/-V'
  CALL lineout RESP_FILE, ctl_line
  /* Call Mountain to backup database files */
  'tape SBK @'RESP_FILE
  call checkrc                      /* Check return code */
end
else if( DB_NAME = 'DBTWO' ) then
do
  rsp_line = linein( RESP_FILE ) /* Read from the response file */

  /******
  /* Only compress the database files. Do not compress the */
  /* the single UIF file which is backed up during MODE 1.  */
  /******

  if( MODE = '2' ) then
  do
    /* ZIP the requested files */
    'PKZIP2 C:\DBTWO\DBTWO.ZIP' 'rsp_line
    call checkrc          /* Check return code from PKZIP */
  end
  else
  do
    /* XCOPY the requested file(s) */
    'XCOPY 'rsp_line' C:\DBTWO'
    call checkrc          /* Check return code from OS/2 */
  end
  end
else
  log_err( "Database name not recognized" )

```

Figure 4: Handling BACKUP for two different databases



```

ARCHIV_PROC:

/* NOTE:  this routine assumes that the input parameters to the  */
/*         User Exit were parsed as follows:                      */
/*         parse upper arg OPT PARM1 PARM2 PARM3 PARM4 PARM5      */

DB_DRIVE = left(strip(PARM1), 1)  /* Strip the colon, ex. C  */
DB_NAME   = strip(PARM2)          /* ex. DBONE          */
LOG_PATH  = strip(PARM3)          /* ex. C:\DBLOGS      */
LOG_FILE  = strip(PARM4)          /* ex. SQL00012.LOG   */

/* Strip the extension and replace it with ZIP */
ZIP_FILE = left(LOG_FILE, 9) 'ZIP'

/* compress the logfile and move it to a holding directory */
'PKZIP2 C:\COLLECT.LOG\'ZIP_FILE LOG_PATH\'LOG_FILE
call checkrc          /* Check return code from PKZIP */
if sqluexitrc <> IBM_RC.OK then
    signal FINISH

/* Archive if log number is divisible by 5 */
LOGNUM = right(PARM4, 1)
if (LOGNUM = '5' | LOGNUM = '0')
do
    'syplus "ArchLogs"
    call checksyrc          /* Check the rc from SYPLUS */

    if sqluexitrc <> IBM_RC.OK then
        do
            /* erase the file, since it will be archived later */
            'ERASE C:\COLLECT.LOG\'ZIP_FILE '1>NUL 2>NUL'
            signal FINISH
        end
    else
        do
            /* clean up - erase files from holding directory */
            'ERASE C:\COLLECT.LOG\*.*'
        end
    end
end

```

Figure 5: Grouping log files before archiving

Figure 8 shows a simple subroutine for the Backup function. It parses the input parameters and calls the NSD.

BACKUP AND RESTORE VS. ROLL-FORWARD RECOVERY

Backup and Restore and roll-forward recovery use the User Exit in different ways due to various nuances in each function.

Frequency of User Exit Use

The first major difference to consider is how and when the User Exit is called. For Backup and Restore, the User Exit is called infrequently and on an application or operator basis. For roll-forward recovery, the User Exit is called frequently, at asynchronous times, and is enabled on a per database basis.



```

BACKUP_PROC:

/* NOTE: this routine assumes that the input parameters to the  */
/*       User Exit were parsed as follows:                      */
/*       parse upper arg OPT PARM1 PARM2 PARM3 PARM4 PARM5      */

DB_DRIVE = strip(PARM1)      /* ex. D:                      */
DB_NAME  = strip(PARM2)      /* ex. DBONE                  */
RESP_FILE = strip(PARM3)     /* ex. D:\SQLDBDIR\SQL00001.BKP */
LABEL    = strip(PARM4)     /* ex. DBONE-684686581        */
MODE     = strip(PARM5)     /* ex. 1                      */

/* Strip timestamp from the generic LABEL */
parse var LABEL . '-' dbm_time

/* Create a label for Mountain */
mtn_label = 'DATABASE DBONE BACKUP : PART 'MODE,
           ' : TIMESTAMP = 'dbm_time

/* Create the command line for Mountain, and put it into  */
/* the response file                                     */
ctl_line = ' /L'mtn_label' /A/D/C/-V'
CALL lineout RESP_FILE, ctl_line

continue = 'YES'

/* Loop until either the Backup is successful, or the  */
/* operator decides to terminate it.                    */
do while (continue = 'YES')

  /* Call Mountain to backup database files */
  'tape SBK @RESP_FILE

  if( rc <> 0 ) then
    do
      say 'The Backup to the Mountain tape device failed'
      say 'The return code was 'rc
      say 'Do you wish to attempt the Backup again (Y/N)'

      pull answer

      do while (answer <> 'Y') & (answer <> 'N')
        say '"Y" or "N" only:'
        pull answer
      end

      if( answer = 'N' ) then          /* Operator cancelled retry */
        continue = 'NO'
      end
    else
      continue = 'NO'                /* Successful backup      */
    end
  end
end

```

Figure 6: Prompting the operator for error handling



This difference has several implications. First, if the User Exit is called for **Backup** or **Restore**, there will always be an application or operator to receive a return code. The application can check the return code and take appropriate action if the function fails. On the other hand, the User Exit calls for roll-forward recovery are asynchronous. An application may be doing a query when a problem arises with a User Exit call; the query should not fail because the User Exit failed. These problems are handled in two ways. If the failure occurs during an **Archive** call, the DBM will take no further action on the call. The User Exit should log the failure and inform the operator immediately. It should then return the failure to the DBM, so the DBM knows not to erase the log file. If the failure occurs during a **Retrieve** call, the DBM will display a message on the screen. In this case, the User Exit does not need to inform the operator unless an immediate retry is desirable or necessary. The only exception is that if the **SQLUEXIT** executable is not found for either **Archive** or **Retrieve**, the DBM will display a message on the screen.

The media used, combined with the way the User Exit is called, may have an effect on system performance. If the User Exit sends data to or from a hard disk, there is little performance impact. While hard disks are fast devices designed for frequent reads and writes, tape devices are much slower and largely used for infrequent reads and writes. Similarly, **Backup** and **Restore** cause a large amount of data to be read, written, and stored, while roll-forward recovery necessitates that only small amounts of data (for example, a log file) be read or written. If the database is modified often enough, and the log file is too small, then it is possible to overrun the tape unit. Tapes also have a shorter read and write life than hard disks. When configuring the log files and selecting a device and tape size for roll-forward recovery, an administrator should estimate how many log files will fit on the tape and determine whether the tape is capable of holding up under the expected wear and tear. The choice of peripheral used for archiving data, tape versus optical disk, and so on, will be in part a function of all the preceding considerations.

Grouped vs. Stand-Alone Calls

A second difference to consider is that the User Exit is called twice in a row for **Backup** and **Restore**. The backed up data should be kept together rather than divided between tapes. If the second call fails, the entire operation fails; if either dataset is lost, the **Restore** will fail. With roll-forward recovery, each call stands alone. Allowing a **Backup** to be divided across media requires that the programmer create a User Exit that is able to handle prompting. The user must then do more media management than merely sequencing tapes. In environments with more than one database backed up to the same device, it is recommended that the programmer ensure that no two database backups are allowed to go to the same tape or intertwine on a device. You can use the **mode** parameter to check for and prevent this condition.

ARGV	MEANING	SAMPLE DATA
argv[0]	User Exit executable	SQLUEXIT.EXE
argv[1]	Function requested	BACKUP
argv[2]	Database drive	C:
argv[3]	Database name	MYDBASE
argv[4]	Response file	C:\SQLDBDIR\SQL00001.BKP
argv[5]	Label	MYDBASE-684686581
argv[6]	Mode	1

Table 2: Writing the User Exit in C: interpretation of ARGV array

Automatic Truncation of Small Log Files

A third difference is that, for roll-forward recovery, log files are truncated and archived whenever all database use stops (that is, whenever no **Start Using Database** is in effect for that database). Even though the primary log file may be configured to hold 100,000 bytes, it may be archived when it contains only 30,000. Frequent small modifications, with gaps between them, could cause a small log file to be archived after each modification. This would certainly overrun some tape devices and could affect system performance regardless. To prevent overrun, try a **Start Using Database** in an OS/2 session, perhaps using the DBM command-line processor, and



```

int      rc;                /* return codes from system calls */
unsigned char cmd[256];     /* buffer for building OS/2
                           command strings */

int  main( int argc, char *argv[ ] )
{
    if( argc != 7 ) /* expecting seven arguments to be passed */
    {
        exit( FAIL_PARM );
    }

    log_parms( argv );      /* Write input parameters to a
                           /* history run file */

    /******
    /* Call appropriate routine for function requested
    /******

    if( strcmp(argv[1], "BACKUP") == 0 )
        backup( argv );      /* BACKUP function */
    else if( strcmp(argv[1], "RESTORE") == 0 )
        restore( argv );     /* RESTORE function */
    else if( strcmp(argv[1], "ARCHIVE") == 0 )
        archive( argv );     /* ARCHIVE function */
    else if( strcmp(argv[1], "RETRIEVE") == 0 )
        retrieve( argv );     /* RETRIEVE function */
    else
    {
        /* Write the error to the run file, and exit */
        log_err("Invalid command option:", argv[1]);
        exit( FAIL_OPTION );
    }

    return( 0 );
}

```

Figure 7: Writing the User Exit in C: The `main` routine

leave it open throughout the day. This precaution prevents small logs from being frequently archived.

Using the User Exit

A fourth major difference is how DBM is instructed to use the User Exit. For **Backup** or **Restore**, the DBM will call the User Exit if the drive specification is zero rather than a letter. This is the case whether or not the call is made from within an application program, via an API, or from DBA Tools, which use the APIs. Thus, some **Backups** can be directed to use the User Exit. For roll-forward recovery, on the

other hand, DBM will always call the User Exit if the database is *configured* to be recoverable. It is not possible to specify archiving some log files and not others.

SUMMARY

User Exit is a powerful, flexible feature that gives database users the ability to use storage devices not currently supported by the OS/2 operating system. It is advantageous for both large and small databases and can support both **Backup** and **Restore** and roll-forward recovery.



```

/* Small coding example for Backup function using OS/2 XCOPY */

void backup( char *parms[ ] )
{
/*****
/*      Expected parameters for BACKUP function:      */
/*      */
/*      parms[0] : User Exit executable : "SQLUEXIT.EXE" */
/*      parms[1] : Function requested   : "BACKUP"      */
/*      parms[2] : Database drive       : "C:"          */
/*      parms[3] : Database name        : "MYDBASE"     */
/*      parms[4] : Response file        : "C:\SQLDBDIR\SQL00001.BKP"*/
/*      parms[5] : Label                : "MYDBASE-684686581" */
/*      parms[6] : Mode                 : "1"           */
/*      */
*****/

    unsigned char db_drive[3];           /* database drive */
    unsigned char db_name[9];           /* database name (alias if assigned) */
    unsigned char resp_file[25];        /* response file */
    unsigned char label[20];            /* backup label */
    unsigned char mode[2];              /* backup mode */

    FILE *rf;                          /* response file handle */
    unsigned char rsp_line[81];         /* response file line buffer */

    strcpy( db_drive, parms[2] );        /* Most of these are not used in */
    strcpy( db_name, parms[3] );         /* this code segment but are shown */
    strcpy( resp_file, parms[4] );        /* to complete the example.      */
    strcpy( label, parms[5] );
    strcpy( mode, parms[6] );

/*****
/* Open the response file provided by Database Manager which contains */
/* the list of files to be backed up. */
/* */
/* The Database Manager requires two calls to SQLUEXIT to complete a */
/* database backup process. The first call (MODE parameter = 1) request */
/* SQLUEXIT to back up the special "UIF" file. The response file would */
/* contain a single line of the form: */
/*      C:\SQLUTIL\MYDBASE.UIF */
/* */
/* The second and final call (MODE = 2) request SQLUEXIT to backup the */
/* directory containing the actual database. The response file would */
/* contain a single line of the form: */
/*      C:\SQL00001\*. */
*****/

    if( (rf = fopen(resp_file, "r")) == NULL ) /* open response file */

```

Figure 8: Writing the User Exit in C: the Backup routine (continued on page 120)



```

{
    fprintf( run, "\nFailed to open response file: %s\n", resp_file );
    exit( FAIL_OPEN_RESPONSE );
}

fgets( rsp_line, 81, rf );          /* get line from response file */
fclose( rf );                      /* close response file */

/*****
/* Build the XCOPY command line string to pass to OS/2 */
*****/

strcpy( cmd, "XCOPY ");
strcat( cmd, rsp_line );
strcat( cmd, " C:\\SQLLIB\\DB_BACKS\\" );
strcat( cmd, db_name );
fprintf( run, "Backup command = <%s>\n", cmd );

rc = system( cmd ); /*pass command to OS/2 and let it do the work */

if( rc != 0 )
{
    fprintf( run, "\nFailed system call: %s\n", cmd );
    fprintf( run, " Return Code = %d\n", rc );
    exit( FAIL_OS2 );
}

```

Figure 8: Writing the User Exit in C: the Backup routine (continued from page 119)

Dawn Bezviner, IBM Corp., 11400 Burnet Rd., Austin, Texas, 78758. Ms. Bezviner has worked for IBM since 1980. She joined the OS/2 EE Database Manager group in 1988, and is currently assigned to the LAN Systems User Interface Group, and is assisting in the design and development of object-oriented products. She has taught classes at IBM covering database and operating systems. She has a B.A. in mathematics from the University of Pennsylvania and an M.S. in computer science from the University of Texas at Austin.

Richard Lawrence, IBM Corp., 11400 Burnet Rd., Austin, Texas, 78758. Lawrence is a senior associate programmer for the OS/2 EE Database Manager. He joined IBM in 1988 and worked on security for OS/2 EE. He was also responsible for the Backup/Restore User Exit samples shipped with ES 1.0.

REFERENCES

For a full description of Roll Forward Recovery, see *IBM Extended Services for OS/2 Guide to Database Manager*

LAN



The LAN Transport Layer of Extended Services and LAN Services

by Dana Beatty

Before LAN Services (LS) and Extended Services (ES) came to exist, there was OS/2 Extended Edition (EE). Creating LS and ES was not just a matter of splitting EE into two separate packages, but rather a metamorphosis that affected the LAN Transport layer of these two services.

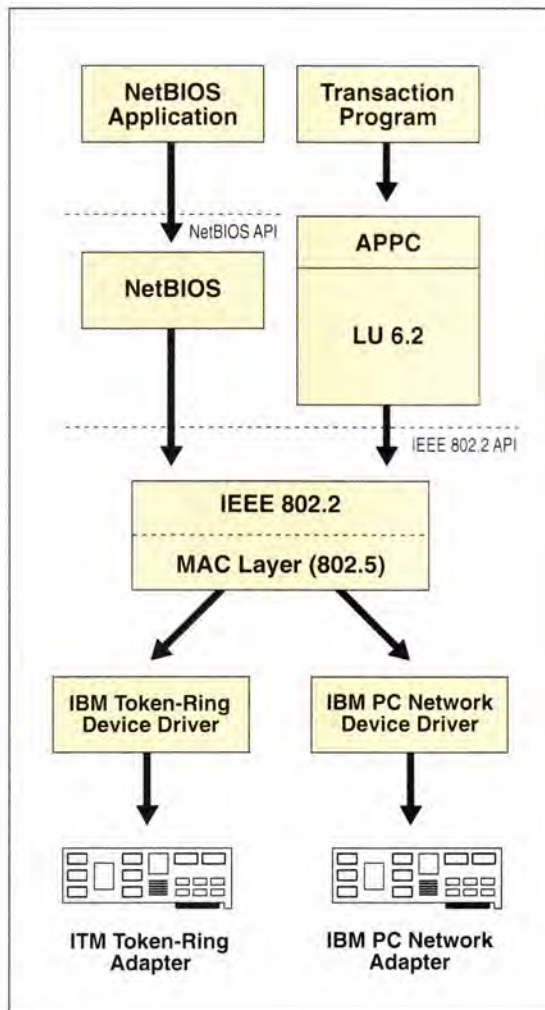


Figure 1: OS/2 Extended Edition LAN protocol stack

IN THE BEGINNING

In the beginning, there was OS/2 Standard Edition (SE), OS/2 Extended Edition (EE), and IBM's LAN Server. SE provided the operating system, while EE contained the LAN protocol stacks (NetBIOS and IEEE 802.2), some selected LAN adapter card device driver support, and the OS/2 requester support. LAN Server contained the server and DOS LAN Requester functions.

At the time, there was a logical division of functions between the three software packages. SE provided an operating system base for its PS/2 line of systems that the customer could build upon, adding options like EE. EE provided a cohesive set of protocols, mainly geared towards host connectivity, and included LAN protocols that allowed APPC to operate over LANs. Finally, LAN Server could be added to provide server function for local area networks.

In addition to the LAN protocol stack, EE also contained the requester function. The obscure separation of LAN function between EE and LAN Server was hard for customers to understand and necessitated purchase of all three software packages (SE, EE, and LAN Server) just to operate in a LAN-only environment. It also left the bulk of EE potential untapped.

OS/2 was initially designed to support only IBM software and hardware. SE, a prerequisite for EE, also supported only IBM. Even EE initially supported only IBM LAN adapters (IBM Token-Ring Network and IBM PC Network), not adding Ethernet support until version 1.2. Finally, LAN Server was the only



Dana Beatty



choice for server support. Figure 1 shows the OS/2 EE LAN protocol stack.

MAKING OS/2 A MORE OPEN SYSTEM

As OS/2 entered the marketplace with this system, however, customers were beginning to embrace the concept of open systems, demanding interoperability from the hardware and software of different manufacturers. The computer industry quickly moved away from closed systems with proprietary hardware, software, and protocols to shops with a variety of hardware and software. For OS/2 to succeed and thrive in the future, it had to rise to the challenge of interoperability.

Four major changes were made to bring OS/2 closer to the open-systems concept. First was the notion that IBM's SE need not be the only OS/2 base on which services could operate. If another vendor's OS/2 version supported Compaq hardware, then ES and LS had to operate in the Compaq environment.

The second change incorporated the industry standard Network Driver Interface specification (NDIS) into the LAN protocol stack, or LAN transport layer, common to ES and LS. NDIS provides a means for the higher layer protocol support in ES and LS to operate over OEM adapter cards, as shown in Figure 2.

The third change was the announcement of IBM's sale and support of Novell software. As IBM embraced Novell, customers were given a choice of servers.

For OS/2 to succeed and thrive in the future, it had to rise to the challenge of interoperability.

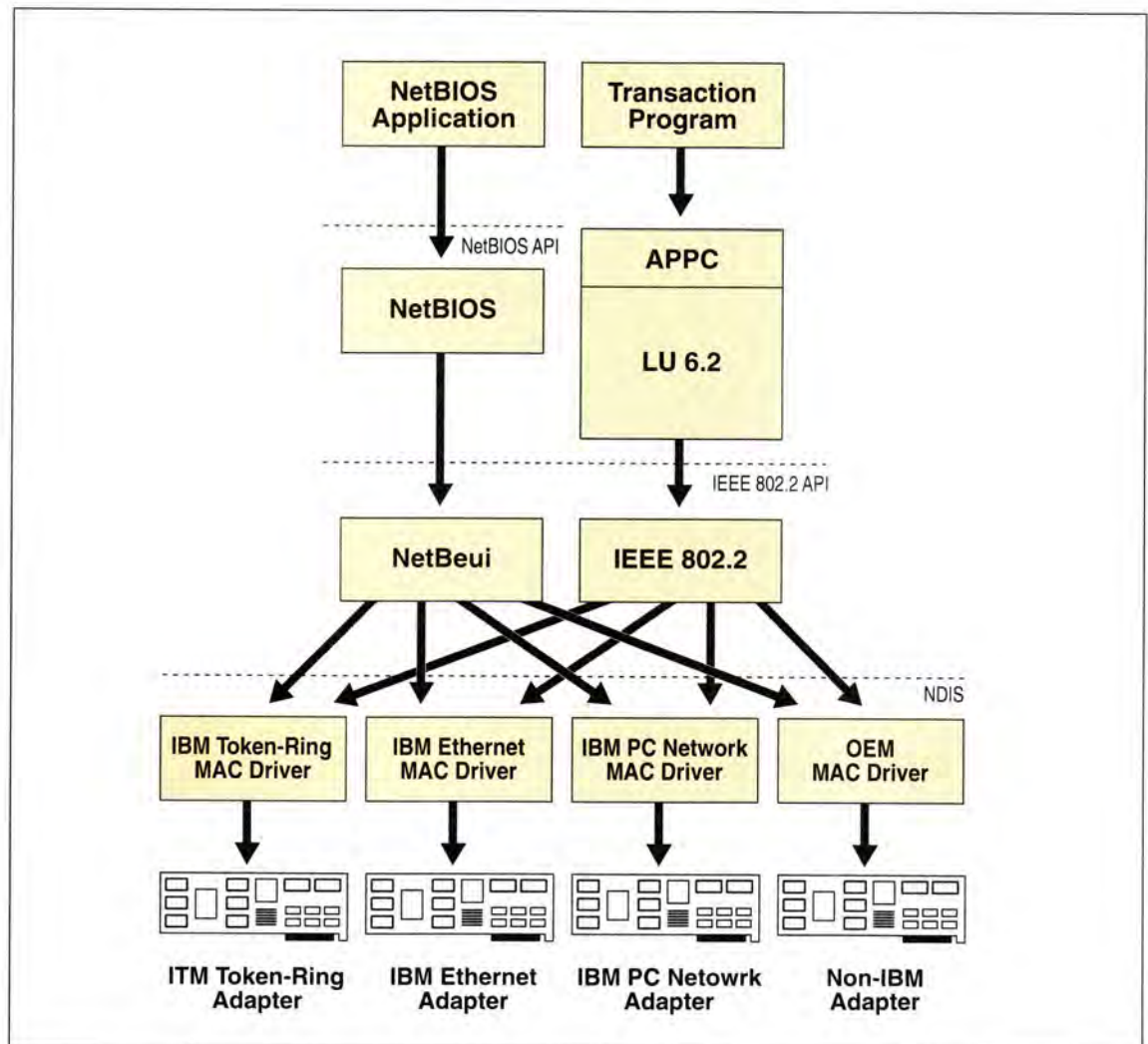


Figure 2: Extended Services and LAN Services LAN transport design



A last change was the fact that the LAN Transport support and requestor function was shipped with ES and LS. This new packaging scheme relieved customers with LAN-only environments of having to purchase three separate products; LAN Services contains all the support they need. Additionally, customers can now choose to purchase either Entry LAN Services or an Advanced LAN Services, which includes Entry LAN Services plus 386 High Performance File System (HPFS) and fault tolerance support.

Entry LAN Services can be installed on either an OS/2 S.E. 1.30.2 or OS/2 2.0 base, while the Advanced LS can only be installed on the OS/2 S.E. 1.30.2 base.

IMPLICATIONS TO EXISTING APPLICATIONS

Although these are substantial changes, they do not affect existing user-written applications. If anything, existing programs should benefit from performance improvements in the new LAN Transport layer of ES and LS. It is important, however, to understand that the LAN Transport layer provided in LS 2.0 and Extended Services v.1.4 cannot coexist with *any* EE version or with ES v.1.3.

SUMMARY

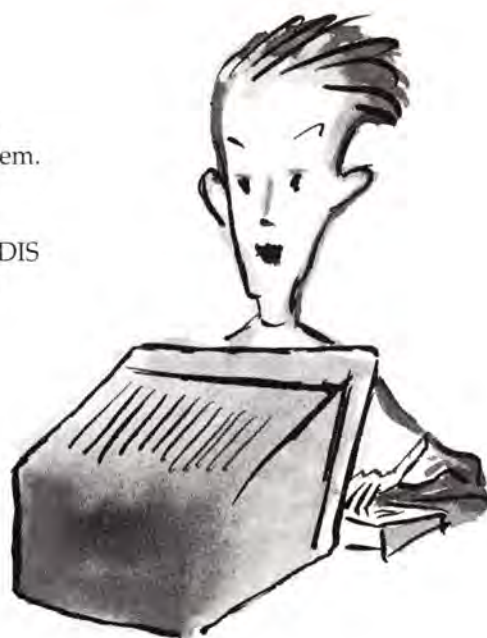
The LAN Transport layer of ES and LAN Services facilitates a more open OS/2 system. When the IBM-only SE prerequisite was removed, these platforms became free to operate on non-IBM systems, while the NDIS support allows ES and LS to operate over vendor adapter cards.

Dana L. Beatty, IBM, 1000 N.W. 51st St., Boca Raton, Fla. 33432. She is a staff programmer in the Communications Subsystem Development I department for IBM Entry Systems Technology in Boca Raton, Fla. She joined IBM in 1985 and worked on the design and development of applications written to the IEEE 802.2 API. She has written magazine articles for IBM Personal Systems Technical Solutions and the IBM Personal Systems Developer, has instructed system engineers at the Arthur K. Watson International Education Center, and has presented at SHARE and GUIDE conferences. She received her B.A. in computer science from the University of Texas, Austin.

REFERENCES

Yarebi, M., K. Bigler, and E. Nelson.
"Support for Ethernet and IEEE 802.2 LAN Protocols," *OS/2 Notebook: The Best of the IBM Personal Systems Developer*. Redmond, Wash.: Microsoft Press, 1990. (IBM Doc. G632-0003)

Network Driver Interface Specification, available from Microsoft Corp. in Redmond, Wash., or 3COM Corp. in Santa Clara, Calif.





LAN

Creating 32 Bit NetBIOS C Programs



Dana Beatty

by Dana Beatty

This article assists developers in creating an executable version of the NetBIOS program found on the LAN Technical Reference Sample Program Listings disk.

THE NECESSARY INGREDIENTS

To ensure that your system has all the necessary software ingredients, you may have to access the IBM Extended Services (ES) for OS/2 2.0 or Lan Services (LS) and the OS/2 2.0 Toolkit. It is assumed that the following hardware and software has been installed:

Files in SAMPLOS2 directory

NETSAMPO.C	Sample NetBIOS program source code
NETERROR.H	NetBIOS error codes and message mappings
NETGBLV.H	NetBIOS sample program global variables

Figure 1: Files in SAMPLOS2 directory

- The base operating system OS/2 2.0 plus either LS or ES with Communications Manager
- The LAN transport API data structures
- The OS/2 2.0 Programmer's Toolkit
- An OS/2 supported LAN adapter card, such as the IBM Token-Ring Network 16/4 Adapter/A™ or the IBM PS/2 Adapter/A for Ethernet Networks™

- The 32-bit IBM C Set/2 Compiler™.

Obtaining the Sample Programs

To program to either the NetBIOS or IEEE 802.2 application programming interface (API) in OS/2, you will need the *LAN Technical Reference* manual, available separately from OS/2. It contains programming hints, a complete list of commands, and the control block layout for commands. Included with the manual is the *Local Area Network Technical Reference Sample Program Listings* disk containing sample dynamic link routine (DLR) source code for the NetBIOS and IEEE 802.2 APIs. While these programs are offered without warranty, they provide a good base for learning how to program to these APIs. Figure 1 lists the NetBIOS files on this disk.

API Data Structures

After acquiring the *LAN Technical Reference*, make sure the LAN transport API data structures (NetBIOS and IEEE 802.2) have been installed on your system. (Refer to the appendix for details on loading them.) API data structures are available in C, Pascal, and assembly language. Figure 2 lists the C versions of these structures.

For OS/2 LS 2.0, these data structures appear in the IBMCOM\INCLUDE directory.

Toolkit

The OS/2 2.0 Programmer's Toolkit contains libraries and files that can simplify your programming tasks. For example, the sample programs on the *LAN Technical Reference* disk rely on the OS2.H file located in the Toolkit.



NetBIOS C Programming-Language API Data Structures

NETB_1_C.H	Command and return code defines
NETB_2_C.H	Control block data declarations
NETB_3_C.H	Trace data declarations
NETB_4_C.H	NetBIOS dynamic link routine entry point declaration

IEEE 802.2 C Programming-Language API Data Structures

LAN_1_C.H	Command and return code defines
LAN_2_C.H	Status and error return code defines
LAN_3_C.H	Dynamic link routine general control block data declarations
LAN_4_C.H	Dynamic link routine and device driver (DD) data declarations
LAN_5_C.H	Trace data declarations
LAN_6_C.H	IEEE 802.2 dynamic link routine entry point declaration
LAN_7_C.H	TYPE declarations

Figure 2: LAN Transport API "C" programming language data structures

```

PATH=C:\IBMC\BIN;%PATH%
set LIB=C:\IBMC\LIB;c:\OS2;c:\cmLib;%LIB%
set INCLUDE=C:\IBMC\INCLUDE;c:\toolkit20\os2h;c:\cmLib;%INCLUDE%
set INCLUDE=C:\IBMC\INCLUDE;c:\toolkit20\os2bin;%INCLUDE%
set TMP=C:\IBMC\TMP
cd \ibmc\samples

```

Figure 3: CSETENV.CMD for OS/2 LAN Services 2.0

Setting Up the INCLUDE Path

Before you can successfully compile the sample programs, the compiler must have access to all ingredients. Instead of copying them into the directory containing the compiler and the source code, it is easier to set the INCLUDE path and LIB path.

When you install the IBM C Set/2 compiler, a file called CSETENV is created. Ensure that this command file sets a path for the LAN transport API data structures and the toolkit include files.

Figure 3 shows a sample command file which, when executed, will set the INCLUDE path so the compiler can locate key files, such as OS/2.H and the LAN API data structures.

COMPILING THE SAMPLE PROGRAM

Once you have acquired all the necessary ingredients, you can create MAKE and DEF files to compile and link the sample programs. Newer releases of the sample program disk should already contain these files. Begin by creating a MAKE file called NETSAMPO.MAK and enter into it the statements shown in Figure 4.

Next, create a link definition file named NETSAMPO.DEF and enter the statements shown in Figure 5. The definition file will resolve program calls to the LAN Transport NetBIOS interface.



```

netsampo.exe : netsampo.obj
LINK
netsampo.obj,netsampo.exe,,,netsampo.def

netsampo.obj : netsampo.c
CC netsampo.c,netsampo.obj;

```

Figure 4: NETSAMPO.MAK file

```

IMPORTS
ACSNETB.NETBIOS

```

Figure 5: NETSAMPO.DEF file

Minor modifications or enhancements can be made to the sample code to provide solutions for real customer situations.

In preparation for compiling the programs, issue the CSETENV command. Then copy the NetBIOS sample programs from the LAN Technical Reference disk to the IBM\CSAMPLES directory.

If you wish to compile the sample program as a 32-bit version, you must take an extra step, editing NETSAMPO.C and inserting “#define E32T016” at the top of the file. This allows the compiler to use 32-bit include files. To compile the program issue the command:

```
NMAKE NETSAMPO.MAK
```

This should compile and link the sample program, creating an executable file named NETSAMPO.EXE.

RUNNING THE SAMPLE PROGRAM

To obtain a list of options for the sample program, enter:

```
NETSAMPO ?
```

To test the program, open two OS/2 sessions with one acting as the receiver (RECV) and one as the sender (SEND).

SUMMARY

With these leads, you can quickly create an executable NetBIOS program from the sample code on the LAN Technical Reference Sample Program Listings disk. Once this is done, experimenting with the NetBIOS API is easy. Minor modifications or enhancements can be made to the sample code to provide solutions for real customer situations.

APPENDIX

Loading the API Data Structures

For IBM LAN Services for OS/2, the API data structures should automatically be loaded into the IBM\COM\INCLUDE directory during installation. If, however, the directory is empty, you may have failed to perform the shutdown as instructed by the installation program. Check to see if another system on your network, such as the server, has these data structures. Otherwise, reinstall LAN Services.

For IBM Extended Services for OS/2, the API data structures are loaded via the “Install Additional Features” panel of Communication Manager.

Dana L. Beatty, IBM, 1000 N.W. 51st St., Boca Raton, Fla. 33432. She is a staff programmer in the Communications Subsystem Development I department for IBM Entry Systems Technology in Boca Raton, Fla. She joined IBM in 1985 and worked on applications written to the IEEE 802.2 API. Beatty has written magazine articles for IBM Personal Systems Technical Solutions and the IBM Personal Systems Developer, has instructed system engineers at the Arthur K. Watson International Education Center, and has presented at SHARE and GUIDE conferences. She received her B.A. in computer science from the University of Texas, Austin.

REFERENCES

Local Area Network Technical Reference
(IBM Doc. SC30-3383)

Communications Manager

Hyperaccess/5 and OS/2 Multitasking Communications



by M. Keith Thompson

HyperACCESS/5™ (HA/5), from Monroe, Mich. based Hilgraeve Inc., provides asynchronous communications for OS/2 and DOS. HA/5 provides developers and users with high-speed file transfers, extensive terminal emulation, built-in modem support, and innovative virus protection. With a powerful scripting language and remote-control capability, HA/5 offers developers automated multitasking communication sessions and an easy support mechanism.

Since 1985, HyperACCESS has been a powerful communications tool for business PCs. The current release supports up to 175 modems and serves many other functions. HA/5's recent update is joined by an upcoming GUI version for Windows and OS/2 2.0.

HA/5 is currently the only full-featured communication program available for OS/2, supporting interfaces such as COM1 through COM4 on ISA machines, COM1 through COM8 on PS/2 models, user-defined interrupts on COM3 and COM4, OS/2 communication devices, and INT14 networked modems. With Hayes' Enhanced Serial Interface™ (ESI) and OS/2, HA/5 can talk to modems at up to 57,600 bits per second.

One of HA/5's chief strengths is its straightforward, quick, and easy-to-use menu structure. The sliding menus allow you to use either the point-and-shoot method or speed keys to access features.

HA/5 works particularly well as a common communication program for DOS and OS/2 users. You can master HA/5 on one platform

and communicate with it or other communications programs on either DOS or OS/2.

The program also provides users of older PC's (8088 and 80286 based) with reliable, high-speed file transfers, allowing the older PCs to work with the new class of V.32bis modems operating at DTE speeds of up to 57,600 bps.

FEATURES

HA/5 offers developers options beyond those traditionally found in asynchronous communication software, fulfilling three communications functions with its mini Bulletin Board System (BBS), Remote Host mode, and E-Mail system. For example, a software company could set up a BBS for its users, who could call in for software updates and technical support. Using the same program for all communications needs eliminates the need to learn multiple programs.

Terminals

HA/5 offers an extensive terminal emulation set. The 17 terminals it emulates, which provide access to almost any host, include:

- VT-52
- VT-100
- VT-102
- VT-220
- VT-320
- TV925
- TV950
- IBM 3101 and 3278
- ANSI
- TTY
- ADM3A
- VIEWPNT
- WANG



All terminal emulators offer support for function and other editing keys, while the TV and IBM 3101 emulations offer protected fields and block mode. HA/5 supports AT&T and Northern Telecom ISDN terminal adapters. Figure 1 shows the HA/5 character mode interface.

Protocols

HA/5 protocols have been independently tested and found to be the fastest and best implemented of all asynchronous communications programs. HA/5's protocols include:

HyperProtocol, the most efficient of the group, offers additional throughput by utilizing an LZW-based software compression routine during file transfer.

HyperPilot

HyperPilot™, HA/5's scripting language, offers over 150 commands that control virtually all the functions and parameters of a communications session. The HA/5 user manual includes a short tutorial and a complete command listing of HyperPilot, a procedural program with a programming structure that closely resembles that of C. The latest version, HA/5 2.1, includes scripts for accessing remote on-line systems such as MCI Mail, CompuServe, GENie, and BIX.

As in a C program, HA/5's scripts reside in a standard text file with an .HP file extension. This allows you to use the editor of your choice to write the scripts. HA/5 also records keystrokes and stores them in a script file for later use. Before it runs the script, HA/5 compiles the source code into machine code that can be more quickly executed.

BBS

HA/5's BBS offers upload and download capabilities. Through passwords, a system administrator controls which functions each user can access. Some users, for example, might only be able to upload files to the host computer, while others could run programs on the host. For added security, HA/5 includes a callback feature that allows the host computer to call a remote user at a predetermined number when the user enters a password. This feature consolidates all long distance telephone charges at one central location.

E-Mail

HA/5's E-Mail system, like its BBS, is a downsized version of a complete product. The E-mail feature allows users to send and receive messages within a user group. Together, E-mail and the BBS compose a practical technical support system. Figure 2 shows BBS options in HA/5.

Extended Hardware Buffering

With OS/2 1.2, IBM and Microsoft added extended buffering to the 16550 UART in



Figure 1: HA/5 character mode interface

- HyperProtocol™ (available in the public domain)
- Zmodem
- Ymodem
- Ymodem G
- Ymodem Batch
- Xmodem
- 1K-Xmodem
- Xmodem CRC
- Xmodem Checksum
- CompuServe Quick B
- Kermit

PS/2s and other high-end PCs. This provided two benefits to users: more reliable high-speed communications and better response time for foreground applications while communications sessions run in the background. While not all communications programs can use the 16550's capability, HyperAccess recognizes the 16550 and turns on its buffering capability, allowing HA/5 to operate more efficiently in the background.

The user sees a decrease in response time during background file transfers with a normal UART due to the workings of serial communication. High-speed serial communication requires a substantial amount of CPU power. Each time a character comes in, the serial port interrupts the CPU with a request for service. With the 16-byte FIFO buffer turned on in the 16550, the CPU gets interrupted only one sixteenth as often as before, leaving more computing power to the foreground application.

Remote Control

HyperACCESS/5 is the only general-purpose communications program to deliver cross-platform compatibility for remote control. If a DOS remote PC with HA/5 calls an OS/2 host PC with the remote host mode, each computer can execute programs on the other. With the remote host, a programmer can control remote PC applications and operating-system software. In addition, HA/5's Learn Mode can watch, decipher, and optimize remote communications sessions as compiled scripts for later use. With its acquisition of KopyKat™, Hilgraeve is the only vendor to offer remote control under OS/2 PM.

A text editor, included with HA/5, offers full-screen editing capabilities with block move and delete features. Additional features include split-screen editing of two files and search and replace. It is also possible to substitute another editor or word processor. Remote system icons for HA/5/PM are shown in Figure 3.

OS/2

While these features are available under DOS as well as OS/2, HA/5 includes extra functionality specific to OS/2. With OS/2's multitasking, users can perform file transfers

as background tasks while doing other work in the foreground. HA/5 for OS/2 also supports multiple communications sessions, allowing users to work on two or more hosts at once.

CPU Utilization

HA/5's aggressiveness setting controls how much CPU time a communications session receives. While OS/2's preemptive multitasking allows applications to vie for processing time, HA/5 optimizes the multitasking environment by adjusting the aggressiveness setting. With a low setting, the foreground application takes priority and utilizes more CPU cycles.



Figure 2: BBS options in HA/5

HA/5 for OS/2 allows a user to start other OS/2 sessions as child processes and run communications detached. Running HA/5 as a detached process, another option, requires less CPU time and memory, but doesn't allow keyboard control or display. Placing automated calls or answering calls are examples of detached usage.

HA/5 uses shared modems on LAN Server and LAN Manager networks. Instead of entering the usual COM1 through COM4 in the setup screen, you enter the server and queue name of a modem located on another



PC. The server and queue name use the standard UNC (Universal Naming Convention) type name \\<server>\<queue>.

The OS/2 version of HA/5 includes an extra diagnose utility. The /D option, represented graphically as a stethoscope on the PM Desktop, collects data to assist Hilgraeve in reproducing any errors or protection violations. If errors occur while HA/5 is running, you copy the information in the DIAGNOSE subdirectory to a disk. Armed with this data, Hilgraeve technical support can help you isolate the problem.

The virus signature file is easily updated from Hilgraeve's BBS or IBM's VIRUS.LST.

CASE STUDIES

Fifield Corp.

Fifield Corp., a Chicago-based commercial property management organization with six field offices across the U.S., launched a project to automate its accounting functions. Fifield discovered that it could save significant worker time and money by providing on-line access to accounting data with HA/5.

Chuck Hinderliter, Fifield's system network coordinator, manages and maintains all network functions. The network includes IBM PS/2 Model 80s for file servers and Model 55s for client PCs. The client PCs run DOS and OS/2 over the OS/2-based 3Com 3+Open network.

Fifield's first goal was to provide, on a server at the hub of its network, a secure centralized accounting database. The company chose a Timberline™ accounting system because of its multitasking capability under OS/2. Before Fifield adopted HA/5, problems arising at the field offices meant that technical support staff from the central office had to go on site to troubleshoot.

With HA/5's remote host access and scripting features, Fifield's remote offices now have automated communications with the central office accounting database. The company reduced their data transmission time from 15 to two hours a week and simplified the file compression process by replacing external file compression utilities with HyperProtocol file compression. Using HA/5's remote control feature, support staff can perform diagnostics, solve problems, and run system backups on remote computers, thus reducing down time and travel expenses.

Countrywide Funding

Countrywide Funding, a full service mortgage banker with 100 offices across the country, needed a simple, easy way to communicate with its 350 PCs. They chose HA/5 due to its powerful scripting language, OS/2 support, and good technical support.



Figure 3: Remote system icons for HA/5/PM

Virus Filter

Proactive virus detection with HyperGuard™ provides an excellent defense from today's numerous viruses. During file transfers, HA/5 filters incoming data for known virus signatures. When a virus is detected, HA/5 displays a warning message and gives you the option to abort the file transfer and delete the existing portion of the file from the disk. To be sure the virus does not cause any trouble, HA/5 writes all zeros to the file before deleting it. HyperGuard, which uses the same set of virus signatures as IBM's Virus Scanning Program™, currently recognizes 315 viruses.

HyperACCESS/5 now handles all of Countrywide's communications from checking electronic mail services to polling branch offices to accessing government agencies. Using HA/5's scripting language, Countrywide created a custom interface to front end all of its communications services for novice users.

Monroe County Pure Waters

The Monroe County Pure Waters agency, which governs local water use in Monroe County, New York, needed to tie together its county-wide computer system. The agency needed to collect data from computers monitoring plant operations located throughout the county and assemble it at the administrative hub.

After evaluating several methods of data collection, Pure Waters chose OS/2 and the HyperACCESS/5 communications program. OS/2's multithreading and background processing combined with HA/5's custom features and ease of use have made data collection simple and fast.

Supervising accountant John Chase used HA/5's script language, HyperPilot, to run file transfers at predesignated times. For the agency's field employees, Chase created a custom system interface. He also developed a corporate BBS with HyperPilot, providing agency users with access to company information.

Pure Waters uses HA/5 to handle remote diagnostics, system updates, and maintenance, eliminating many field support expenses.

HYPERACCESS FOR OS/2 2.0

Hilgraeve is currently testing new versions of HyperAccess for Windows and PM for release later this year. The 32-bit PM version is designed specifically for OS/2 2.0, with tested DTE speeds up to 57,600 bps.

HyperAccess/PM (HA/PM) takes an object oriented approach to communications. While traditional communication programs allow users to choose entries from a dialing directory, HA/PM displays remote systems as selectable icons in a window.

HA/PM uses OS/2 2.0 to execute instructions. With preemptive multitasking, communication intensive tasks operate in the background without noticeably affecting foreground session performance. Tasks are stored as threads instead of message loops, requiring the fewest possible CPU cycles.

By using PM's dynamic data exchange capability, HA/PM can share data with other communications sessions and applications such as databases, spreadsheets, and word processors. For example, a developer can simultaneously capture real time data from an application on a remote computer and analyze the data using another program.

Both of the upcoming GUI versions include Hilgraeve's new Modular Communications Engine (MCE). The MCE provides faster throughput for multiple communications sessions by separating low-level I/O processing from other communications tasks. It services the communications medium in real time while other portions of the program service the display and keyboard. With this new modular technology, developers can focus on creating custom applications.

CONCLUSION

Hilgraeve's HA/5 combined with OS/2 creates an asynchronous communications environment with features such as virus filtering, a scripting language, remote host operation, and electronic mail. HA/5, the only full-featured communications program for OS/2, provides all the functionality of its DOS program in an easy-to-use product.

Keith Thompson is a communications consultant based in Florida. He is a frequent contributor to computer publications.



With preemptive multitasking, communication intensive tasks operate in the background without noticeably affecting foreground session performance.



System Application Architecture

CCL/2: A User Interface Migration Tool for OS/2 2.0



Ruth Anne Taylor

by Ruth Anne Taylor

If you have worked with OS/2 2.0, you have had a chance to use the CUA 91 controls. Have you wondered how these controls can be used in OS/2 1.3 or Windows 3.0 applications? IBM SAA CUA Controls Library/2, or CCL/2, is your answer.

CCL/2 is a productivity tool that embodies the CUA user interface architecture, using a set of dynamic link libraries that contain the controls in 16-bit OS/2 1.3 and Windows 3.0 environments. These libraries may be incorporated directly with the application code, allowing the developer to focus more on the content and quality of the application rather than on CUA conformance.

WHAT IS THE PURPOSE OF CCL/2?

In support of its goals to make OS/2 2.0 the development platform of choice and to make the new CUA architecture pervasive across multiple operating system environments, IBM provides migration support for OS/2 1.3 and Windows 3.0 applications through graphic user interface controls that conform to the CUA 91 architecture. The controls, including the container, file dialogue, font dialogue, notebook, spin button, slider, and value set, are packaged on 3.5- and 5.25-inch disks for OS/2 1.3 and Windows 3.0. CCL/2 controls in these environments work like those in the OS/2 2.0 base system. The spin button, already a part of OS/2 1.3, is provided in CCL/2 only in the Windows 3.0 format.

CCL/2 CONTROLS

In Figure 1, the value set and slider controls are displayed in the left window. The container control is shown in the right window.

Figure 2 shows the notebook, value set, and slider controls.

Container Control

The container control, a key component of the CUA Workplace Model, is used to group related objects for easy access and retrieval and to present objects in various views. Each view generally provides different information about an item.

The icon view presents the container items in an iconic format with text beneath the icons. Generally, an icon represents the item type.

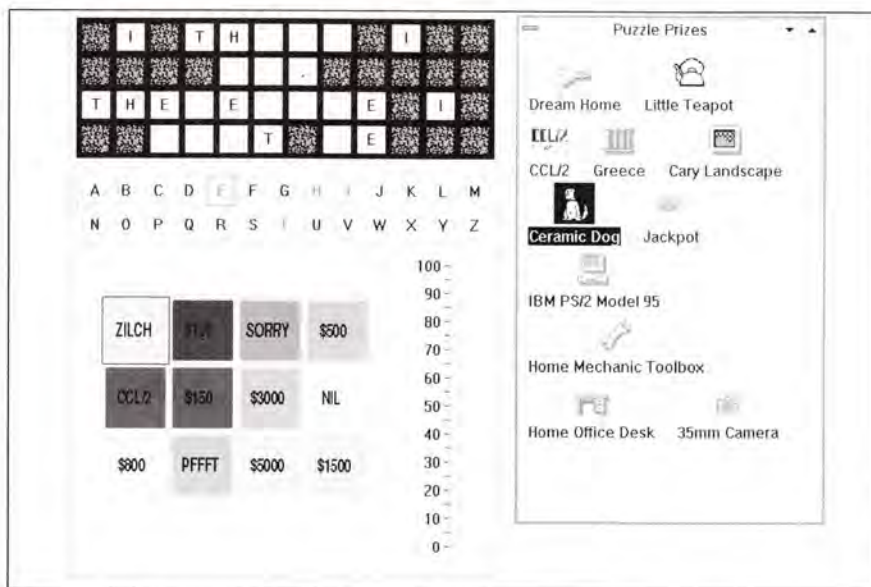


Figure 1: Value set, slider controls, and container control



The name view presents the container items in an iconic format with text to the right of the icons. It can appear either as a single column list or as multiple columns of the same type of data.

The text view presents a simple text list of the contents, and can appear in either single or multiple columns.

The tree view presents a hierarchical view of the container items. Three types of tree views are available: tree icon, tree name, and tree text.

The details view presents information about each container item, arranging information about that item in columns.

File Dialogue

The file dialogue provides a modal dialog that allows applications to consistently perform file operations such as *open* and *save*, implemented as either *Open* or *Save As...* dialogue.

The standard file dialogue includes basic functions that can be extended to meet the requirements of an application. They give users the ability to:

- Display and select from a list of drives, directories, and files
- Enter a file name directly
- Filter the file names before they are displayed
- Display active network connections
- Specify extended attributes for TYPES (for OS/2 Version 1.3 only)
- Interact with a single-selection or multiple-selection file dialogue.

Font Dialogue

The font dialogue provides a dialogue that allows applications to consistently view and select font family names, styles, and available sizes. In the font dialogue, the family name is the name of the typeface. Courier, Times, and Helvetica are some commonly used family names. Type styles include normal, bold, italic, and bold italic. Size is the point size, or vertical

measurement, of the type. Font emphasis styles include underline and strikethrough. The font dialogue provides basic functions that can be extended to meet the requirements of an application. They give users the ability to display and select from a list of:

- Font family names installed on the system
- Styles for each font
- The available sizes for each font
- The emphasis styles available for each font.

In addition, users can view their selections in a preview area, using a sample character string, and can interact with a modal or modeless font dialogue.

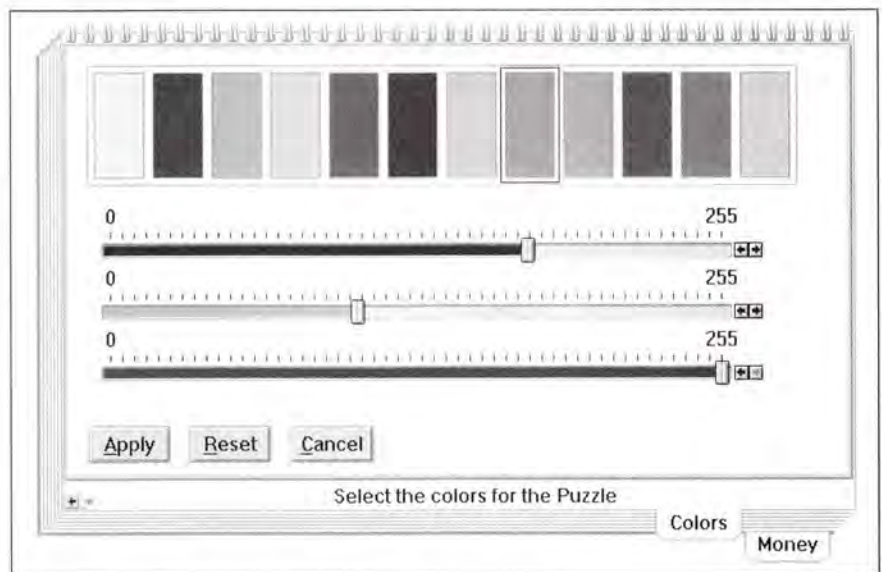


Figure 2: Notebook, value set, and slider controls

Notebook Control

The notebook control simulates an actual notebook, which can be used to organize information in list or calendar format. Related pieces of information can be organized into sections, divided by theme, and represented by text or icon. Sections are marked with tabs, by which the user can navigate from section to section.

Slider Control

The slider control supports the setting of approximate values and properties in analog



rather than digital form, while providing visual feedback on the extent of a value within a range. The slider control emphasizes fluid control and is best used to modify settings rather than to select exact numerical values. In many applications, the slider control can replace awkward use of the scroll bar.

Spin Button Control

The spin button control is used to display one data value at a time by scrolling through a list of available data. The spin button model should reflect real-life objects such as stereo knobs, TV channel knobs, or oven temperature dials. CCL/2 provides the spin button control only for Windows 3.0, as it is already included in OS/2 1.3.

Value Set Control

The value set control allows the user to select one value from a finite set of graphic images supporting the display and selection of choices, which can be represented graphically, numerically, or textually. Choices within a particular value set can also be changed. The graphic images of the value set also preserve screen "real estate." Selecting one choice from a range of alternatives automatically cancels any previously active choice.

Refer to *IBM Personal Systems Developer*, Winter 1992, for detailed articles on the use of each control.

CCL/2 CONTENTS

Sample applications for OS/2 1.3 and Windows 3.0 environments enable application developers to quickly learn the controls. CCL/2 also contains sample application source code and the files necessary to build the applications.

The *Programming Guide*, *Programming Reference for OS/2 Presentation Manager*, and the *Programming Reference for Microsoft Windows* are included in OS/2 1.3 and Windows 3.0 help formats for use online as well as in the manuals. The QuickHelp™ program, included with the Microsoft C compiler, allows developers to access technical information while writing an application.

WHO NEEDS CCL/2?

CCL/2 addresses the user interface development needs of application developers creating programs for OS/2 1.3 and Windows 3.0, and provides tools for conformance to the CUA 91 architecture across these platforms. CCL/2 will offer assistance in migrating these applications to OS/2 2.0 due to the similarity between the controls' programming interfaces and the new CUA APIs in OS/2 2.0. Reusable code that can be shipped directly with the customer's applications increases productivity. Details concerning which portions of CCL/2 can be redistributed with the developed application are specified in the license information shipped in the program package.

PROTECTING INVESTMENT AND STAGING GROWTH

The similarity of CCL/2 to the CUA 91 architecture constructs in OS/2 2.0 increases the opportunity to develop code that can be more easily migrated to OS/2 2.0. Those customers who plan to migrate to OS/2 2.0 with near term OS/2 1.3 or Windows 3.0 development can easily position their application for migration. Because those portions of applications developed for OS/2 1.3 and Windows 3.0 that use CCL/2 will run on OS/2 2.0, organizations can expand technically as their business needs change.

SAVE TIME AND MAKE YOUR JOB EASIER

CCL/2 relieves application developers of significant user-interface development efforts and allows for concentration on competitive functions. The CCL/2 controls simplify conformance to the CUA 91 architecture for OS/2 1.3 and Windows 3.0, while consistent CUA Controls APIs provide easy application migration from OS/2 1.3 and Windows 3.0 to OS/2 2.0.

CCL/2 is designed to allow developers to maximize productivity when developing applications that conform to the CUA 91 architecture. This productivity gain can be realized through reusable components that employ the CUA 91 architecture. Packaged as

CCL/2 relieves application developers of significant user-interface development efforts and allows for concentration on competitive functions.



CHANGING THE VIEW OF A PRESENTATION MANAGER CONTAINER

```
CNRINFO    cnrInfo;

/*****
/* Set the attribute field to the icon view.
*****/
cnrInfo.flWindowAttr = CV_ICON;

/*****
/* Change the view from the current view to the icon view.
*****/
WinSendMsg(
    hwndCnr,                /* Container window handle */
    CM_SETCNRINFO,          /* Container message for setting */
    MPFROMP(&cnrInfo),      /* container control data */
    MPFROMLONG(CMA_FLWINDOWATTR)); /* Message attribute that sets
                                /* container window attributes */
```

CHANGING THE VIEW OF A WINDOWS CONTAINER

```
CNRINFO    cnrInfo;

/*****
/* Set the attribute field to the icon view.
*****/
cnrInfo.flWindowAttr = CV_ICON;

/*****
/* Change the view from the current view to the icon view.
*****/
SendMsg(
    hwndCnr,                /* Container window handle */
    CM_SETCNRINFO,          /* Container message for setting */
    CMA_FLWINDOWATTR,      /* container control data */
    /* Message attribute that sets */
    /* container window attributes */
    (DWORD)(LPVOID)&cnrInfo); /* Container control data */
```

Figure 3: Code samples showing ease of migration

dynamic link libraries, portions of CCL/2 can be shipped directly with the developed application. The License Information shipped in the program package specifies which portions can be redistributed with the developed application.

Additionally, CCL/2 is structured to conform to the same APIs found in OS/2 2.0, which reduces rework in migrating CCL/2 applications from OS/2 1.3 or Windows 3.0 to OS/2 2.0. The APIs for CCL/2 in OS/2 1.3 are

the same as the CUA Controls in OS/2 2.0, but due to differences between OS/2 Presentation Manager and Windows, the APIs for CCL/2 in Windows 3.0 are similar to those in OS/2 2.0.

Figure 3 contains the sample code necessary for changing the view of the container control using OS/2 1.3 Presentation Manager and Windows 3.0.

In CUA 91, objects represented by controls and icons mimic objects in the real world, allowing



the user to move comfortably in the computer environment and to reduce time and errors, thus concentrating on the task at hand. CUA conformance, along with providing the benefits specific to adopting CUA 91 architecture, ensures user interface consistency between applications.

Consistency across applications and environments helps the transfer of knowledge from one product to another, yielding savings in user training. CCL/2 helps stretch this consistency to OS/2 1.3 and Windows 3.0.

CONCLUSION

Using these controls will allow you to develop programs that conform to the application orientation of the CUA 91 architecture for both OS/2 1.3 and Windows 3.0. The consistency of the CCL/2 interfaces with the new APIs in OS/2 2.0 allows user-interface migration with minimal rework.

ACKNOWLEDGMENTS

The CCL/2 product is the work of many people. Special thanks go to Lucy Herman, CCL/2 Business Planner; Phil Kerklo, CCL/2 Product Planner; Steve Robinson, CCL/2 Development Manager; and Jon Holliday for their assistance with this article.



Ruth Anne Taylor, IBM Programming Systems Laboratory, 11000 Regency Parkway, Cary, North Carolina, 27512. Taylor is a staff programmer in the CUA Components Development team. She joined IBM in 1988 and has been working in OS/2 PM development since 1989. She received a B.A. from the University of Kentucky, as well as a B.S. in engineering mathematics and computer science and a Master's of Engineering in computer science, both from the University of Louisville, Kentucky.

REFERENCES

Bernath, David. "File and Font Dialogs: Standardized Selection Techniques," *IBM Personal Systems Developer* (Winter 1992): 18-26.

Bernath, David and Jon Holliday. "Value Set Control: Selecting Graphical Information," *IBM Personal Systems Developer* (Winter 1992): 27-34.

Bernath, David and Jon Holliday. "Slider Control: Slip-Sliding Away in OS/2 2.0," *IBM Personal Systems Developer* (Winter 1992): 35-43.

Brightbill, Pete, Peter Hagggar, Tai Woo Nam, and Ruth Anne Taylor. "Container Control: Implementing the Workplace Model," *IBM Personal Systems Developer* (Winter 1992): 48-54.

Hagggar, Peter. "New Controls in OS/2 2.0: An Overview," *IBM Personal Systems Developer* (Winter 1992): 14-17.

Mack, Diana. "Notebook Control: Organizing, Navigating, and Displaying Data," *IBM Personal Systems Developer*, (Winter 1992): 44-47.

IBM SAA Common User Access Controls Library/2 Programming Guide

IBM SAA Common User Access Controls Library/2 Programming Reference for OS/2 Presentation Manager

IBM SAA Common User Access Controls Library/2 Programming Reference for Microsoft Windows

No Matter What Your GUI Destination **CASEWORKS** Can Take You There



Now You Don't Have to Worry About Choosing the Wrong GUI Development Path...CASEWORKS Lets You Easily Change Direction — by Switching Languages or Platforms — Without Wasting Any Time!

All of the choices surrounding Graphical User Interfaces (GUIs) can make picking a GUI development path seem impossible. Deciding between Windows™ or Presentation Manager™ (PM) is only the first step toward building client/server applications. You need a tool that lets you generate code to Windows or PM for a variety of languages and APIs, and that lets you migrate interfaces among languages and platforms if you change your mind. *CASE:W™ and CASE:PM™ are the only tools that give you that flexibility.*

CASE:W for Windows Development

Windows developers face the challenge of migrating from C to C++ and class library programming. CASE:W makes it easy by generating source code for the three "standards" of Windows programming: the Windows C API, Microsoft Foundation Classes™ and Borland's ObjectWindows™.

Simply buy CASE:W along with a "Knowledgebase" for C, or C++ and a class library. Use the visual designer to create and test your interface. Then generate expert-level, tightly-written, thoroughly-documented code from your knowledgebase. To use the interface in a different environment, simply buy that knowledgebase and generate again. You can even migrate the interface to PM.

CASE:PM for Presentation Manager Development

When you're developing "mission-critical" applications for OS/2™ 2.0, you need a strategic tool that can build PM applications for both 16- and 32-bit PM environments. Our new CASE:PM VIP tool gives you a head start on building client/server applications, and lets you easily migrate legacy applications to OS/2 2.0. Plus, you can leverage all the advanced capabilities of CUA '91.

The Safest Development Route

With CASEWORKS, you get the security of using the leading GUI development tools. Microsoft® includes a version of CASE:W with its QuickC™ for Windows, and more corporations use CASE:PM for developing strategic PM and client/server applications than any other standard language tool. Both tools generate code without restrictions, letting you add any code, anywhere. Plus, a regeneration facility preserves your changes. And *you* control the generated code, with no proprietary languages, runtimes or licensing fees.

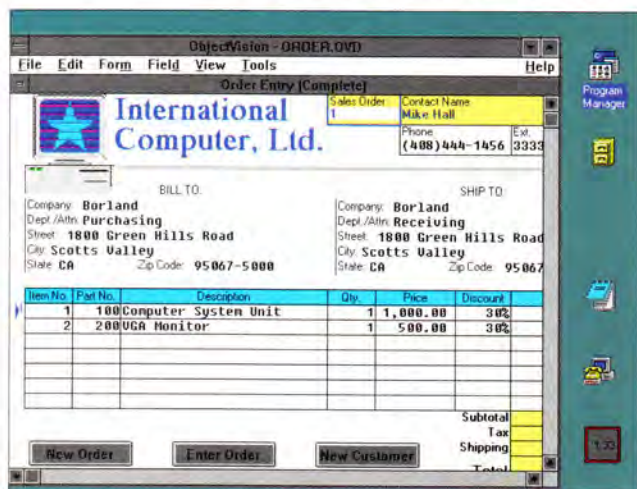
Don't avoid making a decision because you're uncertain which way to go. Choose the only company that can take you to Windows or PM, through any development route: CASEWORKS.

CASEWORKS™

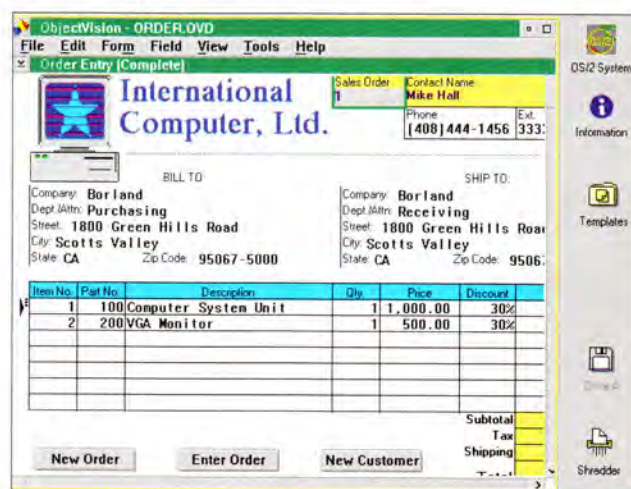
1 Dunwoody Park, Suite 130, Atlanta, GA 30338
1-800-635-1577 • (404) 399-6236 • Fax (404) 399-9516

Q: What do Windows and OS/2 have in common?

New Version



ObjectVision for Windows



ObjectVision for OS/2

A: Your ObjectVision applications.

Borland's ObjectVision[®] brings Microsoft[®] Windows and OS/2[®] users together. Only ObjectVision allows you to write an application *once* for use under both Windows and OS/2. And seamlessly access the same data, regardless of platform. That's because ObjectVision, the world's easiest tool for creating Windows applications, is now also available for OS/2.

Visual programming makes it easy



Tied: #1 Application SoftwareSM
Publisher in Customer Satisfaction[®]

Using revolutionary visual programming techniques, ObjectVision makes it easy for the people who know the business issues to develop and maintain their own graphical software applications without programming.

Creating Windows or OS/2 applications is as easy as A-B-C. Simply:

- A.** Draw your application interface using the convenient form tool.
- B.** Apply your business rules visually.
- C.** Connect your application to your existing database. Or create your own database.

In Windows and OS/2, ObjectVision makes it easy to create, modify and run your own interactive applications. And the OS/2 version gives you easy access to REXX as well as DLLs.

Easy access to data

ObjectVision is the ideal front end to your data because it integrates information from different databases under one familiar, graphical user interface. Your ObjectVision applications can easily connect to information stored in Paradox[®], dBASE[®], Btrieve[®], Database Manager and ASCII formats.

FREE runtime version

Because a runtime version is included *free*, it's easy to distribute your customized applications throughout your company. With ObjectVision, fully interoperable Windows and OS/2 applications are here today!

See your dealer today to get your own copy of ObjectVision for Windows or OS/2, OR call **1-800-331-0877, ext. 6601** to request your ObjectVision for Windows

FREE TRIAL DISK

In Canada, call **1-800-461-3327**



B O R L A N D
The Leader in Object-Oriented Programming

6362-0001-14

